PRILIMINARY SPECIFICATION (REV 0.2)

KS32C6400

32 - BIT RISC

EMBEDDED MICROCONTROLLER

USER'S MANUAL

SAMSUNG ELECTRONICS

SAMSUNG ELECTRONICS CO., LTD.

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible error or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any paricular purpose, nor does Samsung assume any liability arising out of the appication or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such uninteded or unatuorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expensed, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product.

KS32C6400 32-BIT RISC EMMBEDDED MICROCONTROLLER
USER'S MANUAL, REVISION 0.2
Publication Number: 21-32-6400-0298

© 1998 Samsung Electronics

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electric or mechanical, by photocopying, recording, or otherwise, without the prior written consent of Samsung Electronics.

Samsung Electronics' microcontroller business has been awarded full ISO-14001 certification (BVQI Certificate No. 9330). All semiconductor products are designed and manufactured in accordance with the highest quality standards and objectives.

Samsung Electronics Co.,Ltd.
San #24 Nongseo-Lee, Kiheung-Eup
Yongin-City Kyungi-Do, Korea
C.P.O. Box #37, Suwon 449-900

TEL : (0331) 209-2824
FAX : (0331) 209-2839
Home-Page URL: Http://www.samsungsemi.com/

Printed in the Republic of Korea

Preface

The KS32C6400 RISC Microcontroller User's Manual is intended for application designers and programmers who are using the KS32C6400 microcontroller for product development.

The first three sections of this manual give you a general orientation to the KS32C6400 microcontroller:

- Section 1 describes "Overview"
- Section 2 describes "Programmer's Model"
- Section 3 describes "Instruction Set"

Section 4 through 15 describes the special functions of the KS32C6400

Section 4 describes "System Manager"

Section 5 describes "Unified Cache"

Section 6 describes "Derasterizer"

Section 7 describes "Shift Control"

Section 8 describes "Timer"

Section 9 describes "DMA"

Section 10 describes "Parallel Port Interface"

Section 11 describes "UART"

Section 12 describes "Tone Generator"

Section 13 describes " Watch Dog Timer"

Section 14 describes "I/O Ports"

Section 15 describes "Interrupt Controller"

Section 16 describes "LF Motor"

Section 17 describes "CR Control"

Section 18 describes "CR Fire"

Section 19 describes "Print Head"

Section 20 describes "Head DMA"

Section 21 describes "Analog ASIC Interface"

Section 22 describes "RTC"

Section 23 describes "Power Save Control"

Section 24 describes "Electrical Specification"

Section 25 describes " Mechanical Specification"

TABLE OF CONTENTS

SECTION 1 OVERVIEW

Introduction
Features
Block Diagram
Pin Assignments
Signal Descriptions
Pin Type Descriptions
Special Function Registers
ARM7TDMI CPU CORE

SECTION 2 PROGRAMMER'S MODEL

Processor Operating States
Switching State
Memory Formats
Instruction Length
Data Types
Operating Modes
Registers
The Program Status Registers
Exceptions
Interrupt Latencies
Reset

SECTION 3 INSTRUCTION SET

Arm Instruction Set
Introduction Set summary
The Condition Field
Branch and Exchange(BX)
Branch and Branch with Link(B,BL)
Data Processing
PSR Transfer(MRS,MSR)
Multiply and Multiply-Accumulate(MUL, MLA)
Multiply Long and Multiply-Accumulate Long(MULL, MLAL)
Single Data Transfer(LDR, STR)
Halfword and Signed Data Transfer
Block Data Transfer(LDM, STM)
Single Data Swap(SWP)
Software Interrupt((SWI)
Coprocessor Data Operation(CDP)
Coprocessor Data Transfers(LDC, STC)
Coprocessor Register Transfers(MRC, MCR)
Undefined Instruction
Instruction Set Examples
Thumb Instruction Set
Format 1: move shifted register
Format 2: add/subtract
Format 3: mov/compare/add/subtract immediate
Format 4: ALU operations
Format 5: Hi register operation/branch exchange

Format 6: PC-relative load
Format 7: load/store with register offset
Format 8: load/store sign-extended byte/halfword
Format 9: load/store with immediate offset
Format 10: load/store halfword
Format 11: SP-relative load/store
Format 12: load address
Format 13: add offset to Stack Pointer
Format 14: push/pop registers
Format 15: multiple load/store
Format 16: conditional branch
Format 17: software interrupt
Format 18: unconditional branch
Format 19: long branch with link
Instruction Set Examples

SECTION 4 SYSTEM MANAGER

Introduction
System Manager Registers
System Register Address Configuration Register(SYSCFG)
ROM Control Register
SRAM Control Register
DRAM Control Register
DRAM Refresh Control Register
Extra Bank Control Register
Memory Mapping for External Memory I/O
Timing Diagram

SECTION 5 UNIFIED CACHE

Introduction
Cache Operation
Cache Control Register
Selected Cache 2K/4K and internal RAM

SECTION 6 DERASTERIZER

Introduction
Rotation

SECTION 7 SHIFT CONTROL

Introduction
Data Reverse Register
Shift Control Register
Shift Count Register
Shift Word Data Register

SECTION 8 TIMER

Introduction
Timer Control Register
Timer Count Value Register

SECTION 9 DMA

Introduction
DMA Operation
Data Transfer Mode

DMA0, DMA1 Control Register
DMA0, DMA1 Source/Destination Address Register
DMA0, DMA1 Transfer Count Register
DMA2 Control Register
DMA2 Source/Destination Address Register
DMA2 Transfer Count Register

SECTION 10 PARALLEL PORT INTERFACE

Introduction
PPIC Operating Modes
Parallel Port Data Register
Parallel Port Status Register
Parallel Port Ack Width Register
Parallel Port Control Register
Parallel Port Interrupt Event Register
Parallel Port Interrupt Pending Register

SECTION 11 UART

Introduction
UART Operation
UART Line Control Register
UART Control Register
UART Status Register
UART Transmit Holding Register
UART Receive Buffer Register
UART Baud Rate Divisor Register
Timing Diagrams

SECTION 12 TONE GENERATOR

Introduction
Tone Generator Data Register

SECTION 13 WATCH DOG TIMER

Introduction
Watch Dog Timer Control Register
Watch Dog Timer Count Register
Watch Dog Timer Operation

SECTION 14 I/O PORTS

Introduction
I/O Port Mode Register
I/O Port Data Register
Test Control Register
EERAM Control Register
JTAG Test-Logic Unit
Extra Output Port Latch Register
Extra Output Port A Register

SECTION 15 INTERRUPT CONTROLLER

Introduction
Interrupt Mode Register
Interrupt Pending Register
Interrupt Mask Register
FIQ Priority and Jump Address Register

FIQ/IRQ Priority and Jump Address Register
IRQ Priority and Jump Address Register
IRQ Offset Address Register
FIQ Offset Address Register
Interrupt Test Register
IRQ Interrupt Jump Address Register
FIQ Interrupt Jump Address Register

SECTION 16 LF MOTOR

Introduction
Line Feed Motor Control Register
Line Feed Motor Phase Control Register
Line Feed Timer Register

SECTION 17 CR CONTROL

Introduction
Carrier Motor Control Register
Basic Timer Base Register
Prestep Timer Base Register
CR State Control Register
CRSREG Control Register

SECTION 18 CR FIRE

Introduction
Position and Fire Control Register
CR Position and Fire Control Register

SECTION 19 PRINT HEAD

Introduction
Print Head Control Register
Fire Enable Timer/Observation Register
Fire Window Timer/Observation Register
Fire Strobe Delay Timer/Observation Register
Pre-Heat Pulse Timer/Observation Register
Pre-Heat Delay Timer/Observation Register
Print Head Observation Register
Td Delay Counter Register
Print Head Data Word Register
Dot Counter Register
Dot Counter Control Observation Register

SECTION 20 HEAD DMA

Introduction
Head DMA Control Register
Head DMA Source Address Register
Head DMA Transfer Count Register
Head DMA Source/Match Address Register

SECTION 21 ANALOG ASIC INTERFACE CONTROL

Introduction
Analog ASIC Special Register

SECTION 22 RTC

Introduction
RTC Control Register
BCDSEC Counter Register
BCDMIN Counter Register
BCDHOUR Counter Register
BCDDAY Counter Register
BCDDATE Counter Register
BCDMON Counter Register
BCDYEAR Counter Register

SECTION 23 POWER SAVE CONTROL

Introduction
Power Save Control Register

SECTION 24 ELECTRICAL SPECIFICATION

Absolute Maximum Ratings
Thermal Characteristics
D.C. Electrical Characteristics

SECTION 25 MECHANICAL SPECIFICATION

Package Dimensions

1 OVERVIEW

INTRODUCTION

SAMSUNG's KS32C6400 16/32-bit RISC micro controller is designed to provide a cost-effective and high performance micro controller solution for inkjet printer and MFP.

An outstanding feature of the KS32C6400 is its CPU core, a 16/32-bit RISC processor (ARM7TDMI™) designed by Advanced RISC Machines, Ltd. The ARM7TDMI™ core is a low-power, general purpose, microprocessor macro-cell that was developed for use in application-specific and custom-specific integrated circuits. Its simple, elegant, and fully static design is particularly suitable for cost-sensitive and power sensitive applications.

The KS32C6400 is developed using ARM7TDMI™ core, 0.5um CMOS standard cells, and memory compiler. Most of the on-chip function blocks were designed using an HDL synthesizer.

The integrated on-chip function that are described in this document include:

- 4KB Instruction/Data cache
- 2KB Internal SRAM
- DMA Controller
- Interrupt Controller
- UART
- Watch Dog Timer
- Timer
- Real Time Clock
- Power Save Control
- I/O port
- Parallel Port Interface
- Print Head Control
- Carrier Motor Control
- Paper Motor Control
- S/W Assistant function(Rotator, Shifter, Reverser)
- Analog ASIC Serial Interface Control

FEATURES

ARCHITECTURE
- Fully 16/32-bit RISC architecture
- Efficient and powerful ARM7TDMI CPU core
- Cost-effective JTAG-based debug solution

SYSTEM MANAGER
- 32M byte address space
- 8/16-bit external bus support for ROM, SRAM, DRAM(Fast Page, EDO) and external I/O
- Programmable access cycle(2 ~ 7 wait cycles)
- Support sleep mode for low power consumption

UNIFIED CACHE & SRAM
- Instruction/Data cache
- Two way set associative cache with 4KB
- LRU (Least Recently Used)
- Four depth write buffer
- 2KB Internal SRAM.

DIRECT MEMORY ACCESS(DMA)
- 3-channel DMAC
- Memory-to-Print Block with decompression
- Memory-to-memory, memory-to-parallel port, parallel-to-memory, UART-to-memory, memory-to-UART, I/O-to-memory, memory-to-I/O data transfers without CPU intervention.
- Initiated by software or external DMA request.
- Increments or decrements source or 8-bit, 16-bit or 32-bit data transfer

INTERRUPT
- 22 interrupt sources (including 3 external interrupt.)
- Normal or fast interrupt modes(IRQ, FIQ)

UART (SIO)
- Two channel UART (Serial I/O) with DMA based or interrupt based operation; supports 5-bit, 6-bit, 7-bit, or 8-bit serial data transmit/receive
- Programmable baud rates
- Infra-red(IR) Tx/Rx support(IrDA)

TIMER
- Three programmable 16-bit timers
- Interval mode operation

WATCH DOG TIMER
- 16-bit timer useful for periodic reset or interrupts

POWER SAVE CONTROL
- Power saving provides that power dissipation of the peripheral decreases in sleep mode.

PARALLEL PORT INTERFACE
- DMA-based or interrupt-based operation
- Supports IEEE Standard 1284 communication modes (Compatibility mode, nibble mode, bytes mode, and ECP mode)
- Supports ECP protocol with or without run-lengthencoding (RLE)
- Automatic handshaking mode for any forward or reverse protocol with software enable/disable

PRINT HEAD CONTROL
- Support both davinci and Babbage printhead
- Printing data and Fire control

CARRIER MOTOR CONTROL
- Support two kind of Motor (DC and Stepper)
- Support Full / Half / Quarter step mode for stepper

CARRIER POSITION AND FIRE CONTROL
- Support fire control upto 2400 dpi with DC motor
- Carrier position interrupt for easy position control

PAPER MOTOR CONTROL
- Support two kind of motor driver(Uni- / Bi- polar)
- Support Full / Half / Quarter step mode

DERASTERIZER / SHIFTER
- 16 x 16 bit rotate by 90/270 degree for raster data rotation
- Reverse 16bit data
- Shift/Rotate 7 half words with selectable direction

ANALOG ASIC INTERFACE CONTROL
- Support interface with Mustang ASIC

REAL TIME CLOCK(RTC)
- 32.768 KHz clock
- The data include second, minute, hour, date, day, month and year

OPERATING FREQUENCY
- Up to 40MHz

PACKAGE
- 160pin QFP

BLOCK DIAGRAM

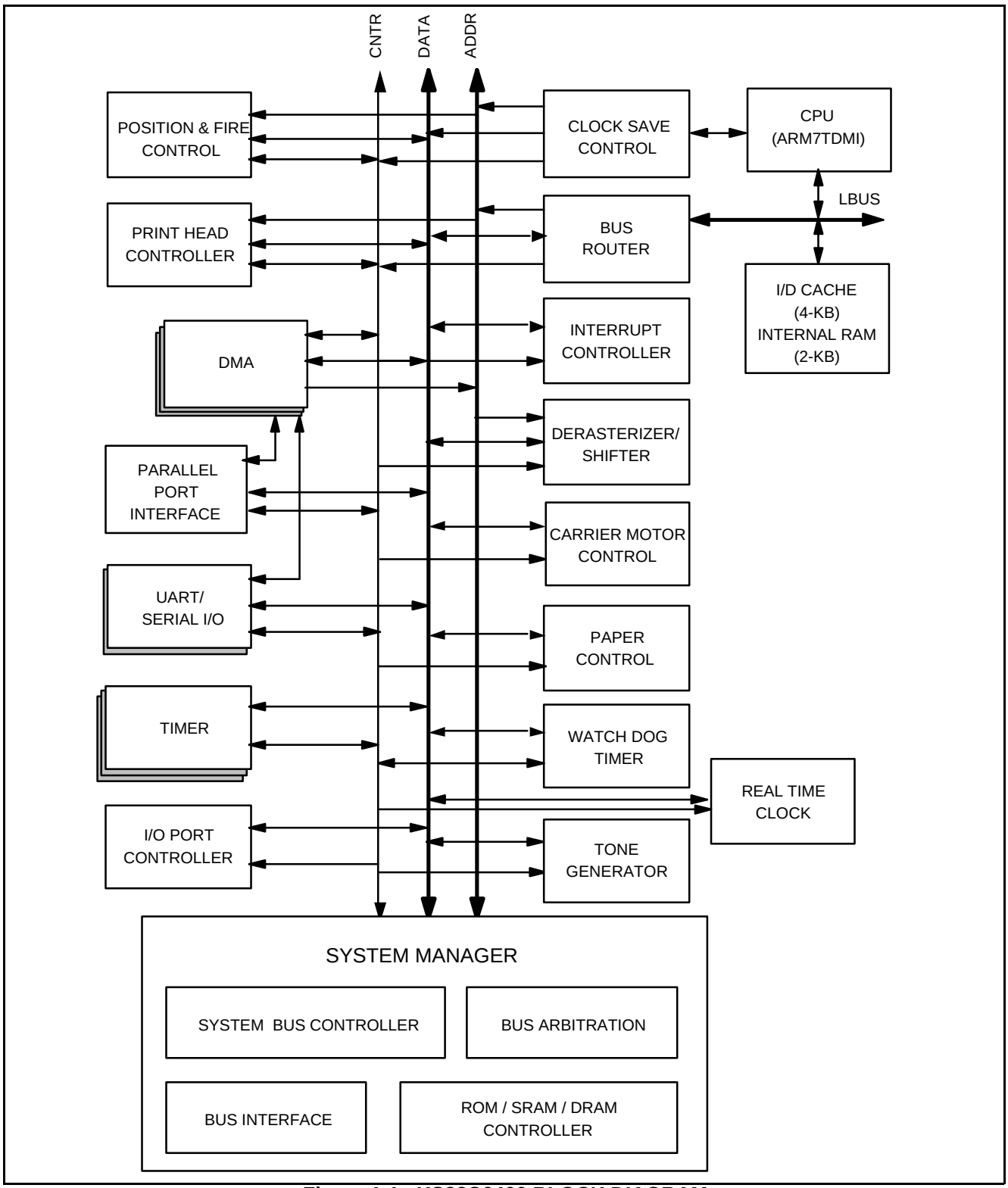


Figure 1-1. KS32C6400 BLOCK DIAGRAM

PIN ASSIGNMENTS

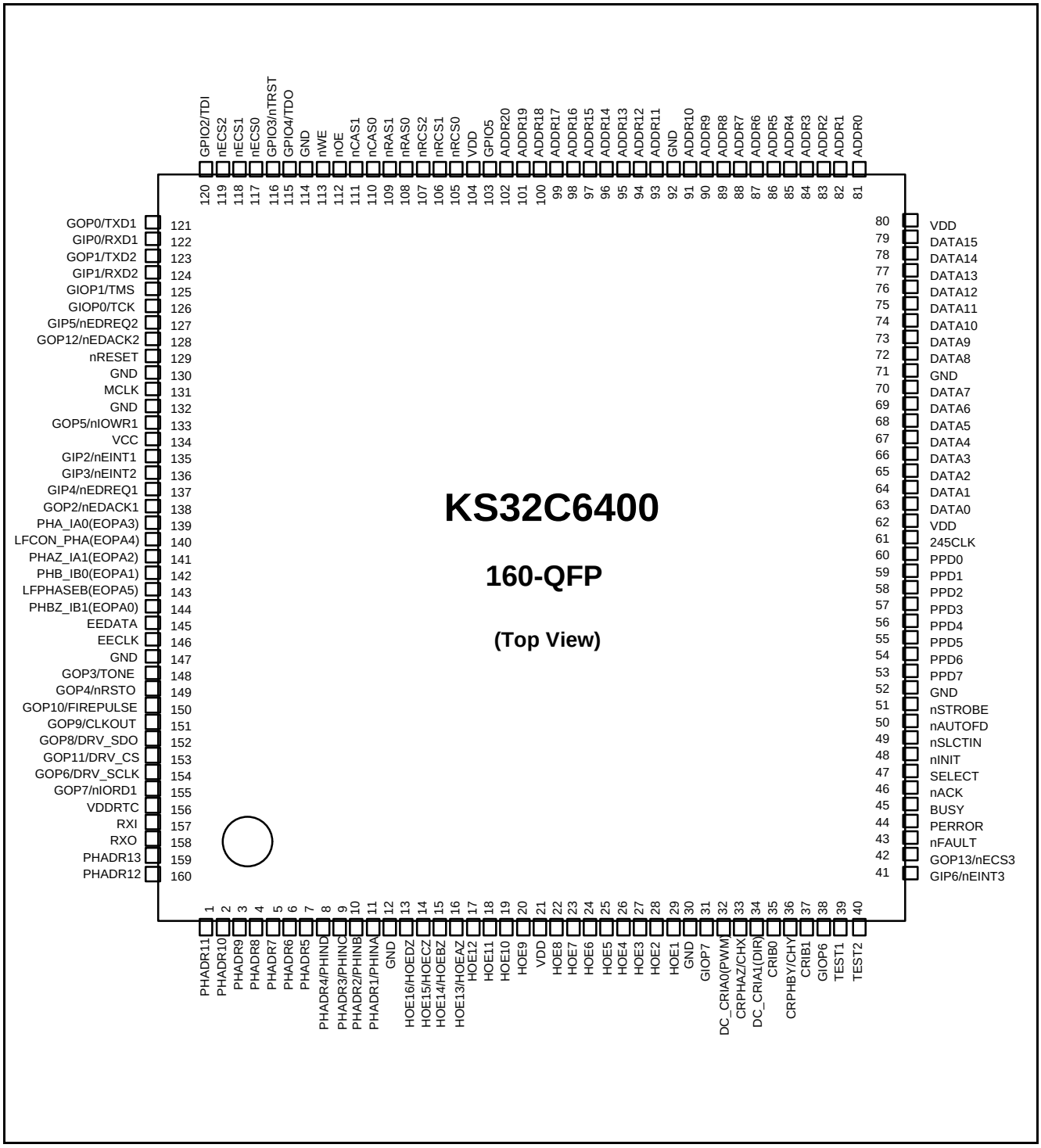


Figure 1-2. KS32C6400 PIN ASSIGNMENTS

PIN DESCRIPTIONS

Table 1-1. KS32C6400 PIN DESCRIPTIONS

SIGNAL	PIN NO.	I/O TYPE	DESCRIPTION
MCLK	131	I2	KS32C6400 master clock. It has a 50% duty cycle and an operating frequency of Max. 40MHz
nRESET	129	I4	Reset. This is the global reset input for the KS32C6400. For a system reset, this pin must be held to low level at least 64 machine cycles.
ADDR[20:0]	81~91, 93~102	I/O3	Address bus. The 21bit address bus, ADDR[20:0], covers the full 2M half-words address range of each ROM/SRAM, DRAM, and external I/O bank
DATA[15:0]	63~70, 72~79	I/O3	External bi-directional 16-bit data bus.
nRAS[1:0]	108,109	O1	Row address strobe for DRAM. The KS32C6400 supports up to two DRAM banks. One nRAS output is provided for each bank.
nCAS[1:0]	110,111	O1	Column address strobe for DRAM. The two nCAS outputs indicate the byte selections whenenver a DRAM bank is accessed.
nOE	112	I/O4	Output enable. Whenever a memory access occurs, the nOE output controls the output enable port of the specific memory device.
nWE	113	O1	Write enable. Whenever a memory access occurs, the nWE output controls the write enalble port of the specific memory device.
nECS[2:0]	119~117	O1	External chip select. Four I/O banks are provided for external memory-mapped I/O operations. Each I/O bank contains up to 2M words. The nECS signals indicate that an external I/O bank is selected.
nRCS2	107	O2	ROM/SRAM chip select. The KS32C6400 can access up to three external ROM/SRAM banks. nRCS[0] corresponds to ROM/SRAM bank 0, nRCS[1] to bank 1, and nRCS[2] to bank 2. By controlling the nRCS signals, CPU addresses can be mapped into the physical memory banks.
nRCS[1:0]	106,105	O1	
nSLCTIN	49	I1	Select information. This input signal is used by parallel port interface to request 'on-line' status information.
nSTROBE	51	I1	Strobe. The nSTROBE input indicates when valid data on parallel port data bus, PPD[7:0]
nAUTOFEED	50	I1	Autofeed. The nAUTOFD input indicates whether data on the parallel port data bus, PPD[7:0], is an autofeed command. Otherwise, the bus signals are interpreted as data only.
nINIT	48	I1	Initialization. The nINIT input signal initializes the parallel port's input control.
nACK	46	O1	Acknowledge for parallel port. The nACK output signal is issued whenever a transfer on the parallel port data bus is completed.

Table 1-1. KS32C6400 PIN DESCRIPTIONS (Continued)

SIGNAL	PIN NO.	I/O TYPE	DESCRIPTION
BUSY	45	O1	Parallel port busy. The BUSY output signal indicates that the KS32C6400 parallel port is currently busy.
SELECT	47	O1	Parallel port select. The SELECT output signal indicates whether the device connected to the KS32C6400 parallel port is 'on-line' or 'off-line'.
PERROR	44	O1	Parallel port paper error. PERROR output indicates that a problem exists with the paper in the inkjep printer. It could indicate that the printer has a paper jam or that the printer is out of paper.
nFAULT	43	O1	Fault. The nFAULT output indicates that an error condition exists with the printer. This signal can be used to indicate that the printer is out of ink or to inform the user that the printer is not turned on.
PPD[7:0]	53~60	I/O1	Parallel port data bus. This 8-bit, tri-state bus is used to exchange data between the KS32C6400 and an external host(peripheral).
245CLK	61	I/O4	Control the direction of the PPD[7:0] at logic test.
PHA_IA0	139	O1	Line feed motor phase signal A.
PHAZ_IA1	141	O1	Line feed motor phase signal AZ.
PHB_IB0	142	O1	Line feed motor phase signal B.
PHBZ_IB1	144	O1	Line feed motor phase signal BZ.
LFCON_PHA	140	O1	Line feed motor control signal 1.
LFPHASEB	143	O1	Line feed motor control signal 2.
CRPHAZ/CHX	33	I/O4	Direction control line for phase A.
CRPHBY/CHY	36	I/O4	Direction control line for phase B.
DC_CRIA0	32	O1	Current control line 0 for phase A.
DC_CRIA1	34	O1	Current control line 1 for phase A.
CRIB0	35	O1	Current control line 0 for phase B.
CRIB1	37	O1	Current control line 1 for phase B.
PHADR[13:1]	159~160, 1~11	O1	Gate control line for print head.
HOE[16:1]	13~20, 22~29	I/O2	Drain control line for print head.
GIOP0/TCK	126	I/O5	JTAG TCK Interface in MDS mode.
GIOP1/TMS	125	I/O5	JTAG TMS Interface in MDS mode.
GIOP2/TDI	120	I/O5	JTAG TDI Interface in MDS mode.
GIOP3/nTRST	116	I/O5	JTAG nTRST Interface in MDS mode.
GIOP4/TDO	115	I/O5	JTAG TDO Interface in MDS mode.
GIOP5	103	I/O6	General I/O port.
GIOP6	38	I/O6	General I/O port.

Table 1-1. KS32C6400 PIN DESCRIPTIONS (Continued)

SIGNAL	PIN NO.	I/O TYPE	DESCRIPTION
GIOP7	31	I/O6	General I/O port.
GIP0/RXD0	122	I4	Receive data input for the UART0. RXD0 is the UART0 channel's input signal for receiving serial data.
GIP1/RXD1	124	I4	Receive data input for the UART1. RXD1 is the UART1 channel's input signal for receiving serial data.
GIP2/nEINT0	135	I3	External interrupt request input nEINT0.
GIP3/nEINT1	136	I3	External interrupt request input nEINT1.
GIP4/ nEDREQ0	137	I3	External DMA0 request.
GIP5/ nEDREQ1	127	I1	External DMA1 request.
GIP6/nEINT2	41	I1	External interrupt request input nEINT2.
GOP0/TXD0	121	O1	Transmit data output for the UART. TXD0 is the UART0 channel's output for transmitting serial data.
GOP1/TXD1	123	O1	Transmit data output for the UART. TXD1 is the UART1 channel's output for transmitting serial data.
GOP2/ nEDACK0	138	O1	DMA0 acknowledge. This active low output signal is generated whenever a DMA0 transfer is completed.
GOP3/TONE	148	O1	Tone generator output.
GOP4/nRSTO	149	O3	Reset out by watch dog timer.
GOP/nIOWR	133	O1	External output write strobe
GOP6	154	O1	Only general output port.
GOP7/nIORD	155	O1	External output read strobe
GOP8	152	O1	Only general output port.
GOP10/ FIREPULSE	150	O1	Fire pulse output.
GOP11/ DRV_CSB	153	O1	Color print head select signal.
GOP12/ nEDACK1	128	O1	DMA1 acknowledge. This active low output signal is generated whenever a DMA1 transfer is completed.
GOP13/ECS3	42	O1	External Memory Chip Select.
EECLK	146	O1	Clock line to EEPROM.(This signal is made by software.)
EEDATA	145	I/O2	Data line form/to EEPROM.(This signal is made by software)
RXI	157	I5	RTC Oscillator clock input.
RXO	158	O4	RTC Oscillator clock output.
VDDRTC	156		RTC Power.
TEST1	39	I2	Test1 pin. At normal operation this pin must be connected to GND.

Table 1-1. KS32C6400 PIN DESCRIPTIONS (Continued)

SIGNAL	PIN NO.	I/O TYPE	DESCRIPTION
TEST2	40	I2	Test2 pin. At normal operation this pin must be connected to GND.
VDD	21, 62, 80, 104, 134		System Power.
GND	12, 30, 52, 71, 92, 114, 130, 132, 147		System Ground.

NOTES.

1. The prefix 'nXXX' in the pin descriptions indicates that the pins are low active signals.
2. The following definitions are used in the pin I/O type descriptions.

PIN TYPE DESCRIPTIONS

Table 1-2. I/O TYPE DESCRIPTION

PIN TYPE	DESCRIPTION
I1	TTL Schmitt trigger level input buffer.
I2	TTL Level input buffer
I3	TTL Schmitt trigger level input buffer with pull-up register.
I4	CMOS Schmitt trigger level input.
I5	Oscillator input.
O1	Normal output buffer.
O2	Tri-State output buffer.
O3	Open drain output buffer.
O4	Oscillator output.
I/O1	TTL Schmitt Trigger Level Input with Pull-up Resistor and Tri-State Output with Medium Slew-Rate
I/O2	TTL Level with Pull-up Resistor and Tri-State Output
I/O3	TTL Schmitt Trigger Level Input with Pull-up Resistor and Tri-State Output with Medium Slew-Rate
I/O4	TTL Schmitt Trigger Level Input with Pull-up Resistor and Tri-State Output
I/O5	TTL Level Input and Tri-State Output
I/O6	TTL Schmitt Trigger Level Input and Tri-State Output

SPECIAL FUNCTION REGISTERS

Table 1-3. Special Function Registers

Group	Register	Offset	R/W	Description	Reset Value
System Manager	SYSCFG	0x0000	R/W	System Register Access Configuration register	0x00001001
	ROMCON	0x3000	R/W	ROM control register	0x80003002
	SRAMCON0	0x3004	R/W	SRAM control register 0	0x00000000
	SRAMCON1	0x3008	R/W	SRAM control register 1	0x00000000
	EXTCON0	0x300C	R/W	I/O bank 0 control register	0x00000000
	EXTCON1	0x3010	R/W	I/O bank 1 control register	0x00000000
	EXTCON2	0x3014	R/W	I/O bank 2 control register	0x00000000
	EXTCON3	0x3018	R/W	I/O bank 3 control register	0x00000000
	DRAMCON0	0x301C	R/W	DRAM control register 0	0x00000000
	DRAMCON1	0x3020	R/W	DRAM control register 1	0x00000000
	REFCON	0x3024	R/W	DRAM refresh control register	0x00000001
Cache	CACHNAB0	0x1000	R/W	Non-cacheable area begin 0 register	0x00000000
	CACHNAE0	0x1400	R/W	Non-cacheable area end 0 register	0x00000000
	CACHNAB1	0x1800	R/W	Non-cacheable area begin 1 register	0x00000000
	CACHNAE1	0x1C00	R/W	Non-cacheable area end 1 register	0x00000000
Derasterizer	DRAST0	0x6000	R/W	Derasterizer data register 0	Undef.
	DRAST1	0x6004	R/W	Derasterizer data register 1	Undef.
	DRAST2	0x6008	R/W	Derasterizer data register 2	Undef.
	DRAST3	0x600C	R/W	Derasterizer data register 3	Undef.
	DRAST4	0x6010	R/W	Derasterizer data register 4	Undef.
	DRAST5	0x6014	R/W	Derasterizer data register 5	Undef.
	DRAST6	0x6018	R/W	Derasterizer data register 6	Undef.
	DRAST7	0x601C	R/W	Derasterizer data register 7	Undef.
	DRAST8	0x6020	R/W	Derasterizer data register 8	Undef.
	DRAST9	0x6024	R/W	Derasterizer data register 9	Undef.
	DRAST10	0x6028	R/W	Derasterizer data register 10	Undef.
	DRAST11	0x602C	R/W	Derasterizer data register 11	Undef.
	DRAST12	0x6030	R/W	Derasterizer data register 12	Undef.
	DRAST13	0x6034	R/W	Derasterizer data register 13	Undef.
	DRAST14	0x6038	R/W	Derasterizer data register 14	Undef.
	DRAST15	0x603C	R/W	Derasterizer data register 15	Undef.

Table 1-3. Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
Shift Control	DATARVS	0x7000	R/W	Data reverser	0x00000000
	SFTCON	0x7004	R/W	Shift control register	0x00000004
	SFTCNT	0x7008	R/W	Shift count register	0x00000000
	SFTDATA0	0x700C	R/W	Shift word data register 0	Undef.
	SFTDATA1	0x7010	R/W	Shift word data register 1	Undef.
	SFTDATA2	0x7014	R/W	Shift word data register 2	Undef.
	SFTDATA3	0x7018	R/W	Shift word data register 3	Undef.
	SFTDATA4	0x701C	R/W	Shift word data register 4	Undef.
	SFTDATA5	0x7020	R/W	Shift word data register 5	Undef.
	SFTDATA6	0x7024	R/W	Shift word data register 6	Undef.
Timer	TCON	0x5800	R/W	System timer control register	0x00000000
	TBCNT0	0x5804	R/W	Timer base/count register 0	Undef.
	TBCNT1	0x5808	R/W	Timer base/count register 1	Undef.
	TBCNT2	0x580C	R/W	Timer base/count register 2	Undef.
DMA	DMACON0	0xC000	R/W	DMA0 control register	0x00000000
	DMASRC0	0xC004	R/W	DMA0 source address register	Undef.
	DMADST0	0xC008	R/W	DMA0 destination address register	Undef.
	DMACNT0	0xC00C	R/W	DMA0 transfer count register	Undef.
	DMACON1	0xD800	R/W	DMA1 control register	0x00000000
	DMASRC1	0xD804	R/W	DMA1 source address register	Undef.
	DMADST1	0xD808	R/W	DMA1 destination address register	Undef.
	DMACNT1	0xD80C	R/W	DMA2 transfer count register	Undef.
	DMACON2	0xC800	R/W	DMA2 control register	0x08000000
	DMASRC2	0xC804	R/W	DMA2 source address register	Undef.
	DMADST2	0xC808	R/W	DMA2 destination address register	Undef.
	DMACNT2	0xC80C	R/W	DMA2 transfer count register	Undef.
Parallel Port	PPDATA	0xB000	R/W	Parallel port data register	0x00000100
	PPSTAT	0xB004	R/W	Parallel port status register	0x000007E8
	PPACKWTH	0xB008	R/W	Parallel port acknowledge width register	Undef.
	PPCON	0xB00C	R/W	Parallel port control register	0x00000000
	PPINTEN	0xB010	R/W	Parallel port enable interrupt event register	0x00000000
	PPINTPND	0xB014	R/W	Parallel port interrupt pending register	0x00000000

Table 1-3. Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
UART	ULCON0	0xE000	R/W	UART channel 0 line control register	0x00000000
	UCON0	0xE004	R/W	UART channel 0 control register	0x00000000
	USTAT0	0xE008	R	UART channel 0 status register	0x000000C0
	UTXBUF0	0xE00C	W	UART channel 0 transmit buffer register	0x00000000
	URXBUF0	0xE010	R	UART channel 0 receive buffer register	0x00000000
	UBRDIV0	0xE014	R/W	Baud-rate divisor register 0	0x00000000
	ULCON1	0xE800	R/W	UART Channel 1 line control register	0x00000000
	UCON1	0xE804	R/W	UART channel 1 control register	0x00000000
	USTAT1	0xE808	R	UART channel 1 status register	0x000000C0
	UTXBUF1	0xE80C	W	UART channel 1 transmit buffer register	0x00000000
	URXBUF1	0xE810	R	UART channel 1 receive buffer register	0x00000000
	UBRDIV1	0xE814	R/W	Baud-rate divisor register 1	0x00000000
Tone Generator	TONDATA	0xF004	R/W	Tone generator data and control register	0x000000FF
Watch-Dog Timer	WTCNT	0xF804	R/W	Watch-dog timer count register	0x00000003
	WTCON	0xF800	R/W	Watch-dog timer control register	0x00000021
I/O Ports	IOPMOD	0x4808	R/W	I/O port mode register	0x00000000
	IOPDATA	0x4804	R/W	I/O port data register	Undef.
	TSTCON	0x4800	R/W	Test control register	0x00000600
	EERAMCON	0x5000	R/W	EERAM control register	0x00000001
	EOPL	0x8000	R/W	Extra-output port latch register	0x00000800
	EOPA	0x8004	R/W	Extra-output port A register	0x000003C0
Interrupt Controller	INTMOD	0x4000	R/W	Interrupt mode register	0x00000000
	INTPND	0x4004	R/W	Interrupt pending register	0x003FFFFF
	INTMSK	0x4008	R/W	Interrupt mask register	0x00000000
	FIQPRI 0	0x400C	R/W	FIQ Interrupt priority & jump address	0x00000000
	FIQPRI 1	0x4010	R/W	FIQ Interrupt priority & jump address	0x00000000
	FIQPRI 2	0x4014	R/W	FIQ Interrupt priority & jump address	0x00000000
	FIPRI 0	0x4018	R/W	FIQ Interrupt priority & jump address	0x00000000
	FIPRI 1	0x401C	R/W	FIQ Interrupt priority & jump address	0x00000000
	FIPRI 2	0x4020	R/W	FIQ Interrupt priority & jump address	0x00000000
	FIPRI 3	0x4024	R/W	FIQ Interrupt priority & jump address	0x00000000
	FIPRI 4	0x4028	R/W	FIQ Interrupt priority & jump address	0x00000000

Table 1-3. Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
	IRQPRI 0	0x402C	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 1	0x4030	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 2	0x4034	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 3	0x4038	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 4	0x403C	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 5	0x4040	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 6	0x4044	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 7	0x4048	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 8	0x404C	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 9	0x4050	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 10	0x4054	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 11	0x4058	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 12	0x405C	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQPRI 13	0x4060	R/W	IRQ Interrupt priority & jump address	0x00000000
	IRQBASE	0x4064	R/W	IRQ base address	0x00000000
	FIQBASE	0x4068	R/W	FIQ base address	0x00000000
	INTTEST	0x406C	R/W	Interrupt generate register for test	0x00000000
	IRQJPADDR	0x4070	R	IRQ Interrupt jump address	0x00000000
	FIQJPADDR	0x4074	R	FIQ Interrupt jump address	0x00000000
LF MOTOR	LFCR	0x8000	R/W	L/F motor control register	0x00000800
	LFPCR	0x8004	R/W	L/F motor phase control register	0x000003C0
	LFTBR	0x8008	R/W	L/F motor phase control register	Undef.
	LFTOR	0x800C	R	L/F motor timer observation register	Undef.
	LFTCBR	0x8010	R/W	L/F motor timer compare base register	0x00000000
	LFTCOR	0x8014	R	L/F motor timer compare obervation register	0x00000000
	LFCON	0x8018	R/W	L/F step each control register	0x00000000
CR CONTROL	CMCR	0x9000	R/W	Carrier motor control register	0x00000004
	BTB1R	0x9004	R/W	Basic timer base register 1	Undef
	BTB2R	0x9008	R/W	Basic timer base register 2	Undef
	PSTBR	0x900C	R/W	Prestep timer base register	Undef
	CRSCR	0x9010	R/W	CR state control register	0x0000E03A

Table 1-3. Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
	CRSREG	0x9030	R/W	CR step each control register	0x00000000
	PWMOBS	0x9014	R	PWM counter observation register	0x00000000
	PWMCYL	0x9018	R/W	PWM cycle timer base register	0x00000000
	PWMONT	0x901C	R/W	PWM on time base register	0x00000000
	ECDTIM	0x9020	R	Encoder counter observation register	0x00000000
	ECDVAL	0x9024	R	Encoder cycle value register	0x00000000
	INTTIM	0x9028	R	Interval counter observation register	0x00000000
	INTVAL	0x902C	R	Interrupt Interval value register	0x00000000
CR FIRE	PFCR	0x9820	R/W	Position & Fire control register	0x00008090
	CPCR	0x9824	R/W	Carrier position counter register	0x00000000
	PSPR	0x9828	R/W	Print start position register	0x0000FFFF
	PSCR	0x982C	R/W	Print slice counter register	0x00000000
	PIR	0x9830	R/W	Position interrupt register	0x0000FFFF
PRINT HEAD	PHCR	0xA000	R/W	Print head control register	0x00000000
	FETR	0xA004	R/W	Fire enable timer register	0x00000000
	FETOR	0xA008	R	Fire enable timer observation register	0x00000000
	FWTR	0xA00C	R/W	Fire window timer register	0x00000000
	FWTOR	0xA010	R	Fire window timer observation register	0x00000000
	FSDTR	0xA014	R/W	Fire strobe delay timer register	0x00000000
	FSDT0OR	0xA018	R	Fire delay strobe timer 0 observation register	0x00000000
	FSDT1OR	0xA01C	R	Fire delay strobe timer 1 observation register	0x00000000
	FSDT2OR	0xA020	R	Fire delay strobe timer 2 observation register	0x00000000
	FSDT3OR	0xA024	R	Fire delay strobe timer 3 observation register	0x00000000
	PHPTR	0xA028	R/W	Pre-heat pulse timer register	0x00000000
	PHPTOR	0xA02C	R	Pre-heat pulse timer observation register	0x00000000
	PHDTR	0xA030	R/W	Pre-heat delay timer register	0x00000000
	PHDTOR	0xA034	R	Pre-heat delay timer observation register	0x00000000
	PHOR	0xA038	R	Print head observation register	0x00000000
	TDCR	0xA03C	R/W	Td delay counter register	0x00000000
	PHDW0R	0xA040	R/W	Print head data word 0 register	0x00000000
	PHDW1R	0xA044	R/W	Print head data word 1 register	0x00000000
	PHDW2R	0xA048	R/W	Print head data word 2 register	0x00000000
	PHDW3R	0xA04C	R/W	Print head data word 3 register	0x00000000

Table 1-3. Special Function Registers (Continued)

Group	Register	Offset	R/W	Description	Reset Value
	PHDW4R	0xA050	R/W	Print head data word 4 register	0x00000000
	PHDW5R	0xA054	R/W	Print head data word 5 register	0x00000000
	PHDW6R	0xA058	R/W	Print head data word 6 register	0x00000000
	PHDW7R	0xA05C	R/W	Print head data word 7 register	0x00000000
	PHDW8R	0xA060	R/W	Print head data word 8 register	0x00000000
	PHDW9R	0xA064	R/W	Print head data word 9 register	0x00000000
	PHDW10R	0xA068	R/W	Print head data word 10 register	0x00000000
	PHDW11R	0xA06C	R/W	Print head data word 11 register	0x00000000
	PHDW12R	0xA070	R/W	Print head data word 12 register	0x00000000
	DCBR	0xA074	R/W	Dot counter black register	0x00000000
	DCYR	0xA078	R/W	Dot counter yellow register	0x00000000
	DCCR	0xA07C	R/W	Dot counter cyan register	0x00000000
	DCMR	0xA080	R/W	Dot counter magenta register	0x00000000
	DCCOR	0xA084	R/W	Dot counter control observation register	0x00000000
HDMA	HDCON	0xD000	R/W	Head DMA control register	0x00000000
	HDSAR	0xD004	R/W	HDMA source address register	Undef
	HDTCR	0xD00C	R/W	HDMA transfer counter register	Undef
	HDSAR0	0xD014	R/W	HDMA source address register 0	Undef
	HDMAR0	0xD018	R/W	HDMA match address register 0	Undef
	HDSAR1	0xD01C	R/W	HDMA source address register 1	Undef
	HDMAR1	0xD020	R/W	HDMA match address register 1	Undef
ASIC	IDRVASIFR	0x5004	R/W	DRV_ASIF data & data output	0x00000000
RTC	RTCCON	0xF840	R/W	RTC control register	0x00000000
	BCDSEC	0xF870	R/W	RTC second register	Undef
	BCDMIN	0xF874	R/W	RTC minute register	Undef
	BCDHOUR	0xF878	R/W	RTC hour register	Undef
	BCDDAY	0xF87C	R/W	RTC day register	Undef
	BCDDATE	0xF880	R/W	RTC date register	Undef
	BCDMON	0xF884	R/W	RTC month register	Undef
	BCDYEAR	0xF888	R/W	RTC year register	Undef
Power Save	PWRSAVCON	0x5820	R/W	Power save control register	0x00000000

ARM7TDMI CPU CORE

INTRODUCTION

The KS32C6400 microcontroller uses the ARM7TDMI processor, designed by Advanced RISC Machines, Ltd., as its CPU core. Samsung's product design offers the advantages of small size, low power consumption, and low price for high-performance devices such as laser beam printers and ink-jet printers.

The ARM7TDMI core is a fully static CMOS implementation. This implementation allows the system clock to be stopped in any part of the cycle with extremely low residual power consumption and no loss of state. The core's architecture is based on Reduced Instruction Set Computer (RISC) principles.

The instruction set and its related decode mechanism are, therefore, much simpler than microprogramming Complex Instruction Set Computer (CISC) systems. This results in a high instruction throughput and impressive real-time interrupt response. The ARM7TDMI has a 32-bit address bus.

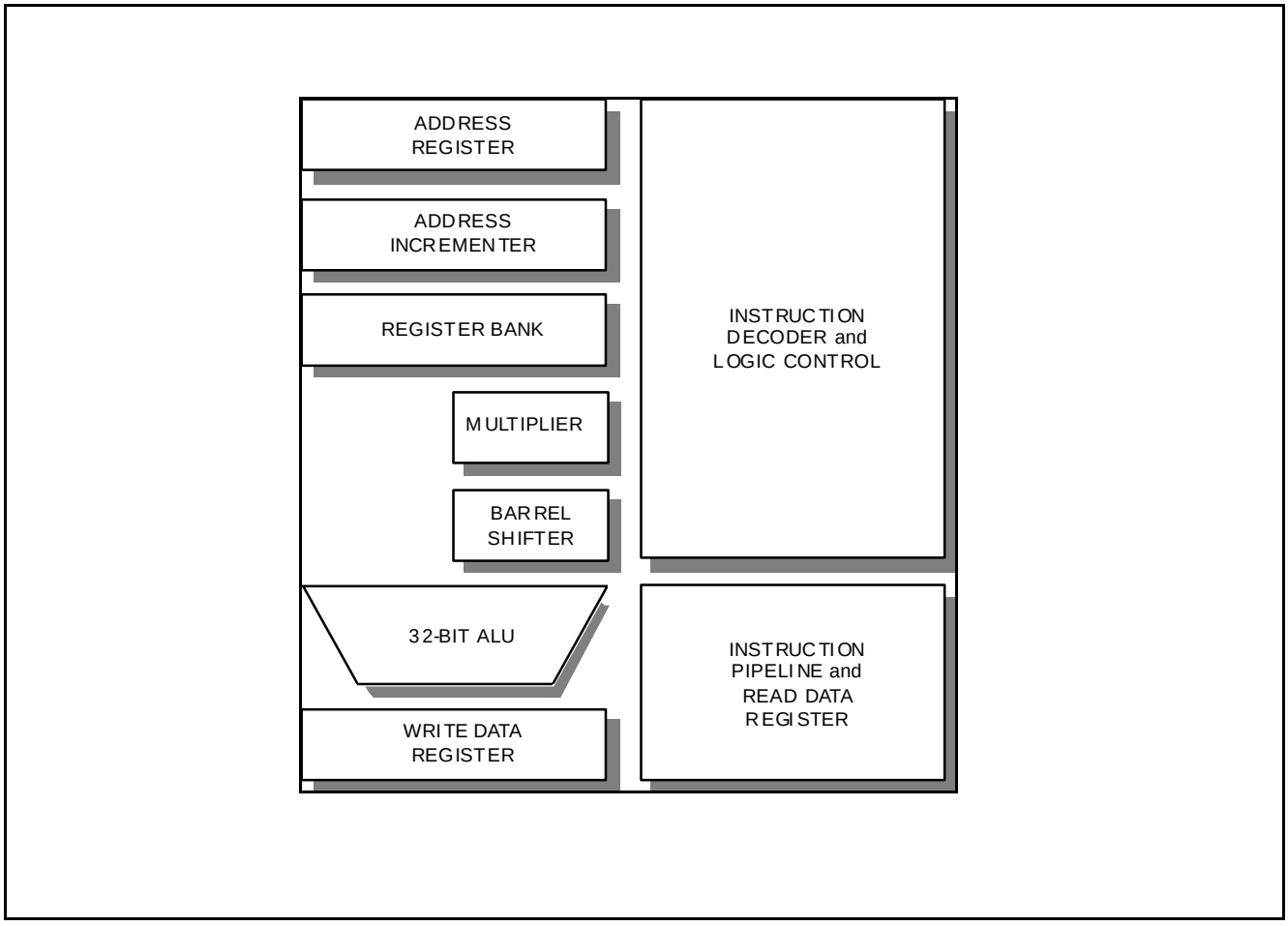


Figure 1-3. ARM7TDMI CPU Core Block Diagram

INSTRUCTION SET

The KS32C6400 instruction set has eleven basic instruction types:

- Two instruction types use the on-chip arithmetic logic unit, barrel shifter, and multiplier to perform high-speed operations on the data in a bank of 31registers, each 32-bit wide.
- Three types control data transfer between memory and the registers. One is optimized for flexibility of addressing, another for rapid context switching, and the third for swapping data.
- Three types control the flow and privilege level of program execution.
- Three types are dedicated to the control of external coprocessors. These instructions extend the off-chip functionality of the instruction set in an open and uniform way.

The ARM instruction set is a good target for compilers of many different high-level languages. Where required for critical code segments, assembly code programming is also straightforward, unlike some RISC processors which depend on sophisticated compiler technology to manage complicated instruction interdependencies.
Pipelining is employed so that all parts of the processor and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.

MEMORY INTERFACE

The CPU memory interface has been designed to allow the performance potential to be realized without incurring high costs in the memory system. Speed- critical control signals are pipelined to allow system control functions to be implemented in standard low-power logic, and these control signals facilitate the exploitation of the fast local access modes offered by industry standard dynamic RAMs.

OPERATING MODES

The CPU core supports a 32-bit data bus and a 32-bit address bus. The data types the processor supports are bytes (8 bits) and words (32 bits), where words must be aligned to four-byte boundaries.
Instructions are exactly one word, and data operations such as ADD are only performed on word quantities. Loads and stores can transfer bytes or words. The CPU supports six operating modes, five of which are visible to the programmer:

- User mode: the normal program execution state
- FIQ (Fast Interrupt Request) mode: designed to support a data transfer or channel process
- IRQ (Interrupt ReQuest) mode: used for general purpose interrupt handling
- Supervisor mode: a protected mode for the operating system

  Undefined mode: entered when an undefined instruction is executed

2 Programmer's Model

INTRODUCTION

KS32C6400 was developed using the advanced ARM7TDMI core designed by Advanced RISC Machines, Ltd.

PROCESSOR OPERATING STATES

From the programmer's point of view, the ARM7TDMI can be in one of two states:

- *ARM state* which executes 32-bit, word-aligned ARM instructions.
- *THUMB state* which operates with 16-bit, halfword-aligned THUMB instructions. In this state, the PC uses bit 1 to select between alternate halfwords.

NOTE

Transition between these two states does not affect the processor mode or the contents of the registers.

SWITCHING STATE

ENTERING THUMB STATE

Entry into THUMB state can be achieved by executing a BX instruction with the state bit (bit 0) set in the operand register.
Transition to THUMB state will also occur automatically on return from an exception (IRQ, FIQ, UNDEF, ABORT, SWI etc.), if the exception was entered with the processor in THUMB state.

ENTERING ARM STATE

Entry into ARM state happens:

1. On execution of the BX instruction with the state bit clear in the operand register.
2. On the processor taking an exception (IRQ, FIQ, RESET, UNDEF, ABORT, SWI etc.). In this case, the PC is placed in the exception mode's link register, and execution commences at the exception's vector address.

MEMORY FORMATS

ARM7TDMI views memory as a linear collection of bytes numbered upwards from zero. Bytes 0 to 3 hold the first stored word, bytes 4 to 7 the second and so on. ARM7TDMI can treat words in memory as being stored either in *Big-Endian* or *Little-Endian* format.

NOTE

The KS32C6400 is configured to the big-endian format.

BIG-ENDIAN FORMAT

In Big-Endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. Byte 0 of the memory system is therefore connected to data lines 31 through 24.

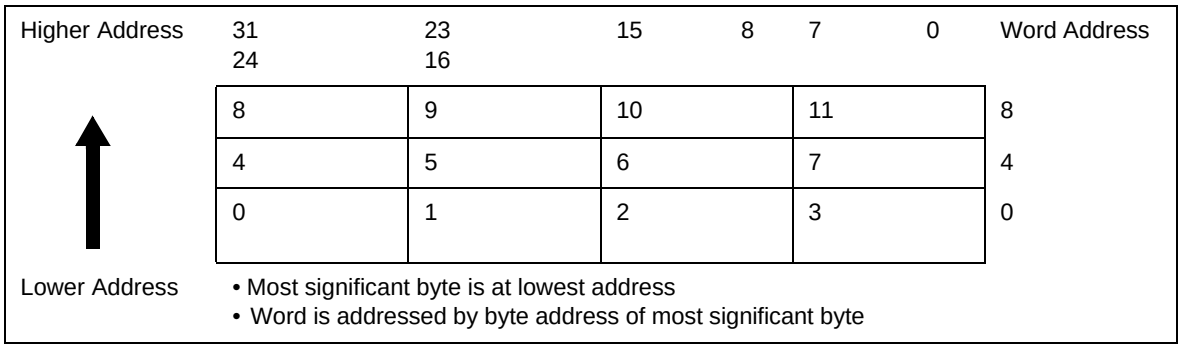


Figure 2-1. Big-Endian Addresses of Bytes within Words

NOTE

The data locations in the external memory are different with Figure 2-1 in the KS32C6400. Please refer to the chapter 4, System Manager.

LITTLE-ENDIAN FORMAT

In Little-Endian format, the lowest numbered byte in a word is considered the word's least significant byte, and the highest numbered byte the most significant. Byte 0 of the memory system is therefore connected to data lines 7 through 0.

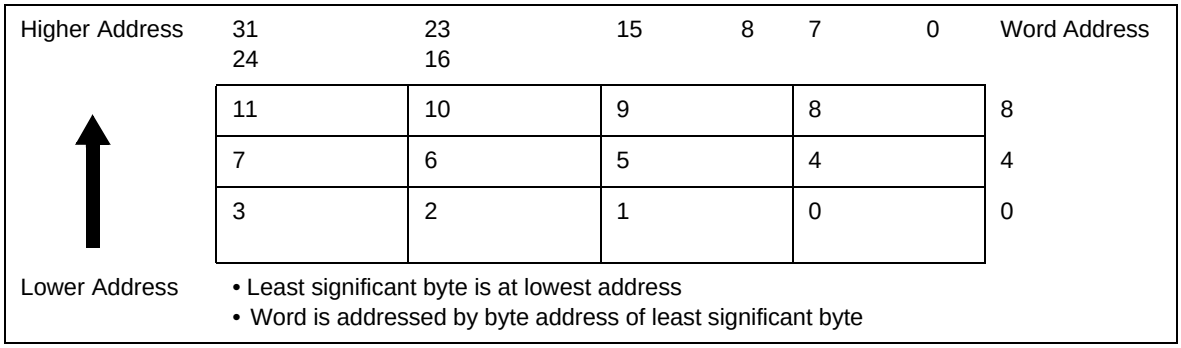


Figure 2-2. Little-Endian Addresses of Bytes within Words

INSTRUCTION LENGTH

Instructions are either 32 bits long (in ARM state) or 16 bits long (in THUMB state).

Data Types

ARM7TDMI supports byte (8-bit), halfword (16-bit) and word (32-bit) data types. Words must be aligned to four-byte boundaries and half words to two-byte boundaries.

OPERATING MODES

ARM7TDMI supports seven modes of operation:

- User (usr): The normal ARM program execution state
- FIQ (fiq): Designed to support a data transfer or channel process
- IRQ (irq): Used for general-purpose interrupt handling
- Supervisor (svc): Protected mode for the operating system
- Abort mode (abt): Entered after a data or instruction prefetch abort
- System (sys): A privileged user mode for the operating system
- Undefined (und): Entered when an undefined instruction is executed

Mode changes may be made under software control, or may be brought about by external interrupts or exception processing. Most application programs will execute in User mode. The non-user modes—known as *privileged modes*—are entered in order to service interrupts or exceptions, or to access protected resources.

REGISTERS

ARM7TDMI has a total of 37 registers - 31 general-purpose 32-bit registers and six status registers - but these can not all be seen at once. The processor state and operating mode dictate which registers are available to the programmer.

The ARM State Register Set

In ARM state, 16 general registers and one or two status registers are visible at any one time. In privileged (non-User) modes, mode-specific banked registers are switched in. Figure 2-3 shows which registers are available in each mode: the banked registers are marked with a shaded triangle.

The ARM state register set contains 16 directly accessible registers: R0 to R15. All of these except R15 are general-purpose, and may be used to hold either data or address values. In addition to these, there is a seventeenth register used to store status information

Register 14 is used as the subroutine link register. This receives a copy of R15 when a Branch and Link (BL) instruction is executed. At all other times it may be treated as a general-purpose register. The corresponding banked registers R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are similarly used to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.

Register 15 holds the Program Counter (PC). In ARM state, bits [1:0] of R15 are zero and bits [31:2] contain the PC. In THUMB state, bit [0] is zero and bits [31:1] contain the PC.

Register 16 is the CPSR (Current Program Status Register). This contains condition code flags and the current mode bits.

FIQ mode has seven banked registers mapped to R8-14 (R8_fiq-R14_fiq). In ARM state, many FIQ handlers do not need to save any registers. User, IRQ, Supervisor, Abort and Undefined each have two banked registers mapped to R13 and R14, allowing each of these modes to have a private stack pointer and link registers.

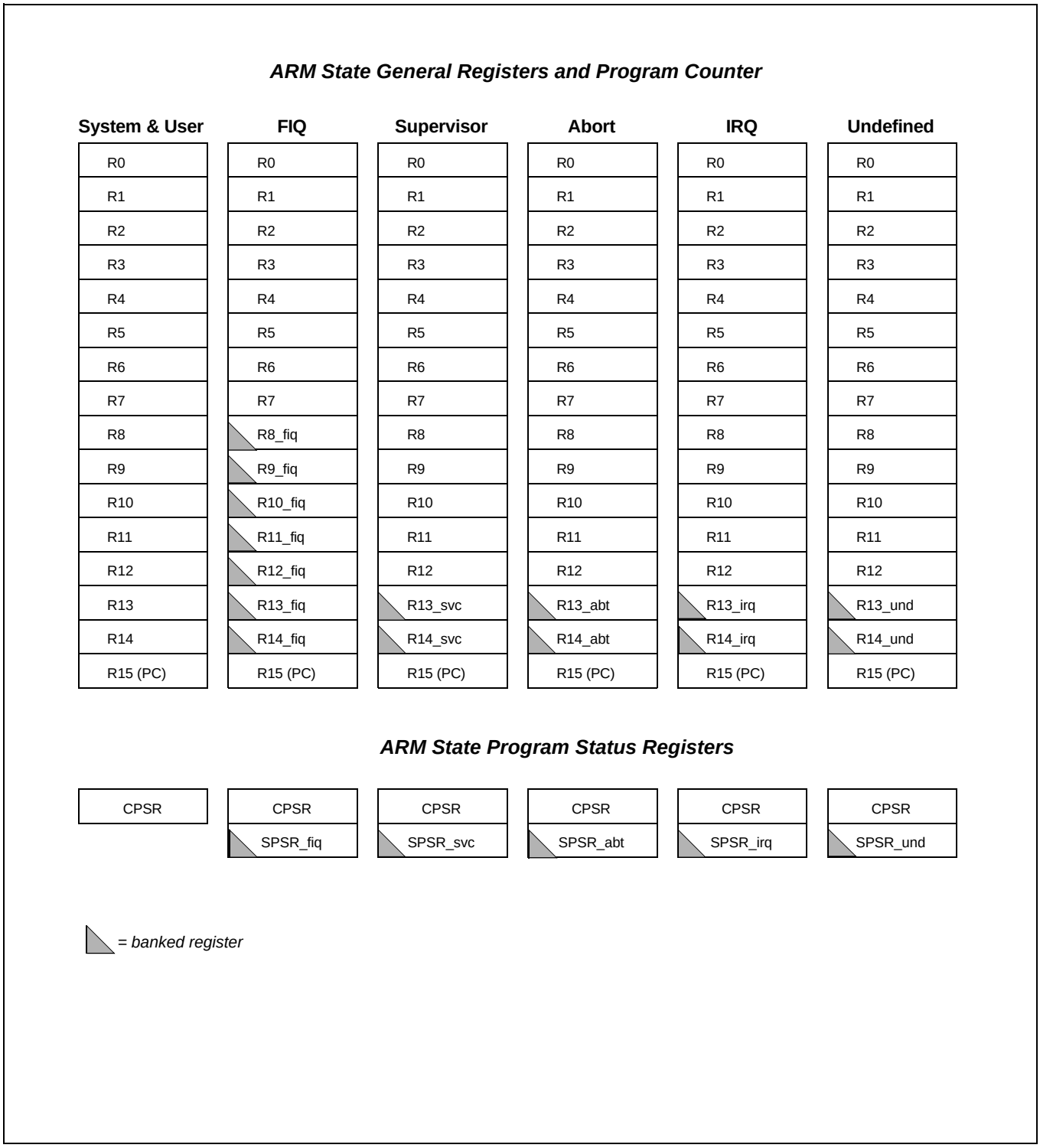
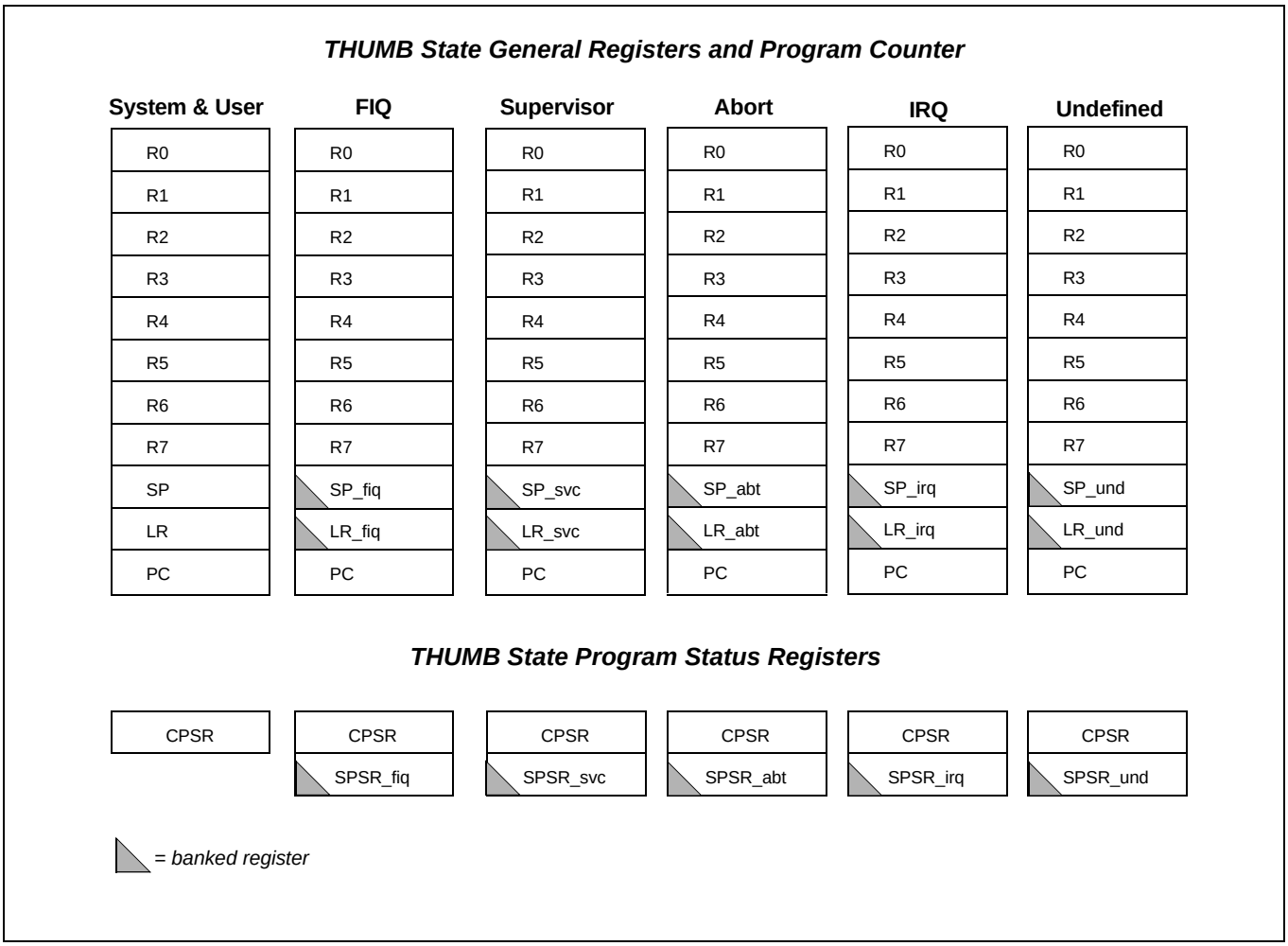


Figure 2-3. Register Organization in ARM State

The THUMB State Register Set

The THUMB state register set is a subset of the ARM state set. The programmer has direct access to eight general registers, R0-R7, as well as the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the CPSR. There are banked Stack Pointers, Link Registers and Saved Process Status Registers (SPSRs) for each privileged mode. This is shown in Figure 2-4.

Figure 2-4. Register Organization in THUMB State



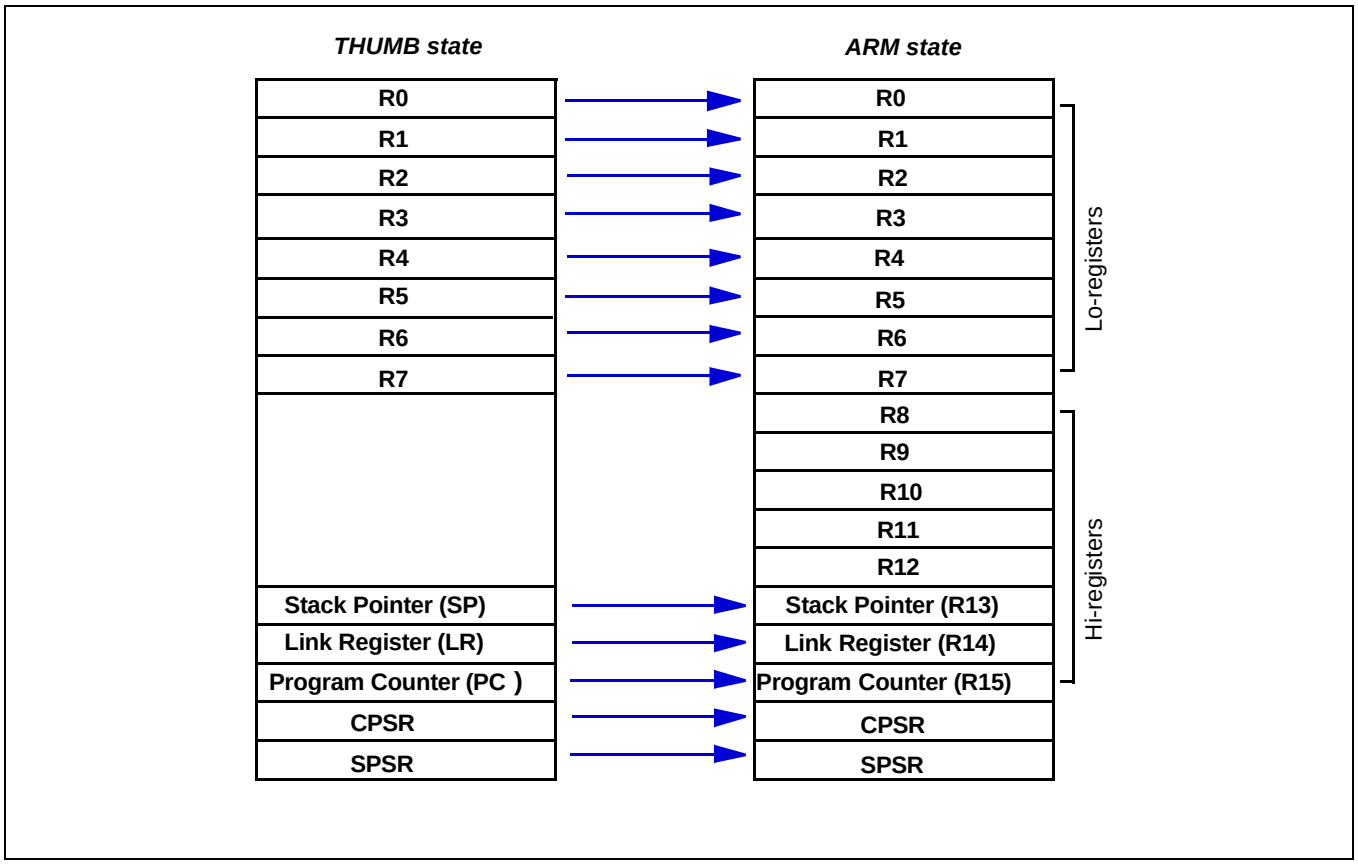
The relationship between ARM and THUMB state registers

The THUMB state registers relate to the ARM state registers in the following way:

- THUMB state R0-R7 and ARM state R0-R7 are identical
- THUMB state CPSR and SPSRs and ARM state CPSR and SPSRs are identical
- THUMB state SP maps onto ARM state R13
- THUMB state LR maps onto ARM state R14
- The THUMB state Program Counter maps onto the ARM state Program Counter (R15)

This relationship is shown in Figure 2-5.

Figure 2-5. Mapping of THUMB State Registers onto ARM State Registers



Accessing Hi-Registers in THUMB State

In THUMB state, registers R8-R15 (the *Hi registers*) are not part of the standard register set. However, the assembly language programmer has limited access to them, and can use them for fast temporary storage.

A value may be transferred from a register in the range R0-R7 (a *Lo register*) to a Hi register, and from a Hi register to a Lo register, using special variants of the MOV instruction. Hi register values can also be compared against or added to Lo register values with the CMP and ADD instructions. For more information, refer to Figure 3-34.

THE PROGRAM STATUS REGISTERS

The ARM7TDMI contains a Current Program Status Register (CPSR), plus five Saved Program Status Registers (SPSRs) for use by exception handlers. These register's functions are:

- Hold information about the most recently performed ALU operation
- Control the enabling and disabling of interrupts
- Set the processor operating mode

The arrangement of bits is shown in Figure 2-6.

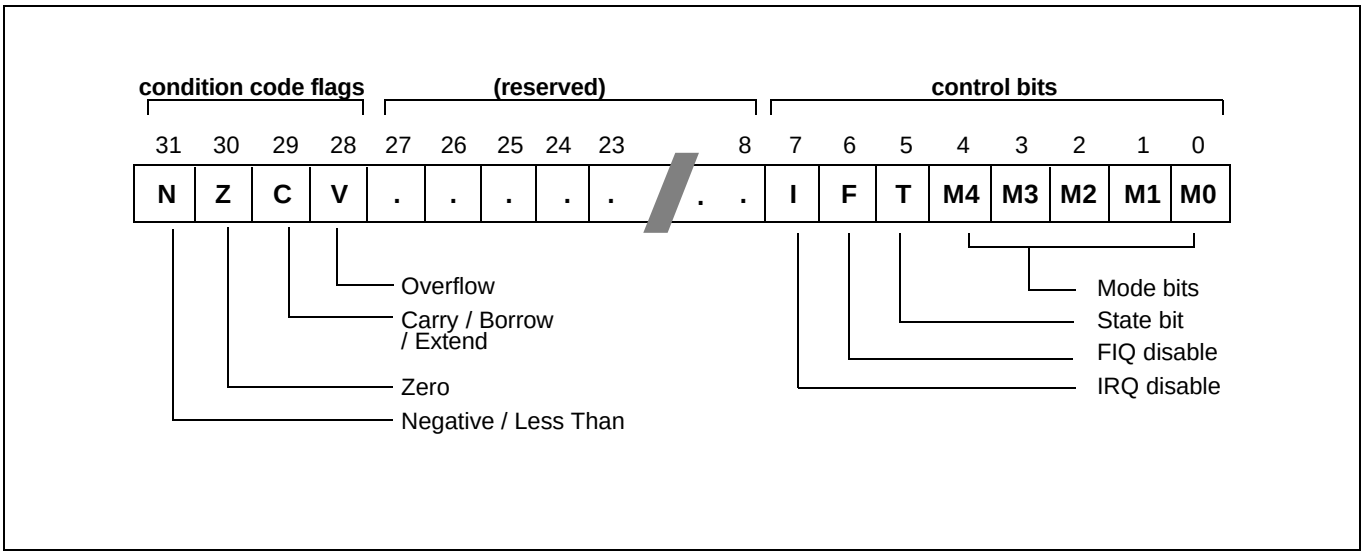

condition code flags
(reserved)
control bits
31 30 29 28 27 26 25 24 23 8 7 6 5 4 3 2 1 0
N Z C V . . . . . . I F T M4 M3 M2 M1 M0
Overflow
Carry / Borrow / Extend
Zero
Negative / Less Than
Mode bits
State bit
FIQ disable
IRQ disable

Figure 2-6 . Program Status Register Format

The Condition Code Flags

The N, Z, C and V bits are the condition code flags. These may be changed as a result of arithmetic and logical operations, and may be tested to determine whether an instruction should be executed.

In ARM state, all instructions may be executed conditionally: see Table 3-2 for details.

In THUMB state, only the Branch instruction is capable of conditional execution: see Figure 3-46 for details.

The Control Bits

The bottom 8 bits of a PSR (incorporating I, F, T and M[4:0]) are known collectively as the control bits. These will change when an exception arises. If the processor is operating in a privileged mode, they can also be manipulated by software.

The T bit — This reflects the operating state. When this bit is set, the processor is executing in THUMB state, otherwise it is executing in ARM state. This is reflected on the **TBIT** external signal.

Note that the software must never change the state of the **TBIT** in the CPSR. If this happens, the processor will enter an unpredictable state.

Interrupt disable bits — The I and F bits are the interrupt disable bits. When set, these disable the IRQ and FIQ interrupts respectively.

The mode bits — The M4, M3, M2, M1 and M0 bits (M[4:0]) are the mode bits. These determine the processor's operating mode, as shown in Table 2-1. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used. The user should be aware that if any illegal value is programmed into the mode bits, M[4:0], then the processor will enter an unrecoverable state. If this occurs, reset should be applied.

Table 2-1. PSR Mode Bit Values

M[4:0]	Mode	Visible THUMB state registers	Visible ARM state registers
10000	User	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR
10001	FIQ	R7..R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7..R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12..R0, R14_irq..R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12..R0, R14_svc..R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12..R0, R14_abt..R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7..R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12..R0, R14_und..R13_und, PC, CPSR
11111	System	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR

Reserved bits

The remaining bits in the PSRs are reserved. When changing a PSR's flag or control bits, you must ensure that these unused bits are not altered. Also, your program should not rely on them containing specific values, since in future processors they may read as one or zero.

EXCEPTIONS

Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. Before an exception can be handled, the current processor state must be preserved so that the original program can resume when the handler routine has finished.

It is possible for several exceptions to arise at the same time. If this happens, they are dealt with in a fixed order. See Exception Priorities on page 2-14.

Action on Entering an Exception

When handling an exception, the ARM7TDMI:

1. Preserves the address of the next instruction in the appropriate Link Register. If the exception has been entered from ARM state, then the address of the next instruction is copied into the Link Register (that is, current PC + 4 or PC + 8 depending on the exception. See Table 2-2 on for details). If the exception has been entered from THUMB state, then the value written into the Link Register is the current PC offset by a value such that the program resumes from the correct place on return from the exception. This means that the exception handler need not determine which state the exception was entered from. For example, in the case of SWI, MOVS PC, R14_svc will always return to the next instruction regardless of whether the SWI was executed in ARM or THUMB state.
2. Copies the CPSR into the appropriate SPSR
3. Forces the CPSR mode bits to a value which depends on the exception
4. Forces the PC to fetch the next instruction from the relevant exception vector

It may also set the interrupt disable flags to prevent otherwise unmanageable nestings of exceptions.

If the processor is in THUMB state when an exception occurs, it will automatically switch into ARM state when the PC is loaded with the exception vector address.

Action on Leaving an Exception

On completion, the exception handler:

1. Moves the Link Register, minus an offset where appropriate, to the PC. (The offset will vary depending on the type of exception.)
2. Copies the SPSR back to the CPSR
3. Clears the interrupt disable flags, if they were set on entry

NOTE

An explicit switch back to THUMB state is never needed, since restoring the CPSR from the SPSR automatically sets the T bit to the value it held immediately prior to the exception.

Exception Entry/Exit Summary

Table 2-2 summarises the PC value preserved in the relevant R14 on exception entry, and the recommended instruction for exiting the exception handler.

Table 2-2. Exception Entry/Exit

	Return Instruction	Previous State		Notes
		ARM R14_x	THUMB R14_x	
BL	MOV PC, R14	PC + 4	PC + 2	1
SWI	MOVS PC, R14_svc	PC + 4	PC + 2	1
UDEF	MOVS PC, R14_und	PC + 4	PC + 2	1
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 4	2
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4	2
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 4	1
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 8	3
RESET	NA	–	–	4

NOTES

1. Where PC is the address of the BL/SWI/Undefined Instruction fetch which had the prefetch abort.
2. Where PC is the address of the instruction which did not get executed since the FIQ or IRQ took priority.
3. Where PC is the address of the Load or Store instruction which generated the data abort.
4. The value saved in R14_svc upon reset is unpredictable.

FIQ

The FIQ (Fast Interrupt Request) exception is designed to support a data transfer or channel process, and in ARM state has sufficient private registers to remove the need for register saving (thus minimising the overhead of context switching).

FIQ is externally generated by taking the **nFIQ** input LOW. This input can except either synchronous or asynchronous transitions, depending on the state of the **ISYNC** input signal. When **ISYNC** is LOW, **nFIQ** and **nIRQ** are considered asynchronous, and a cycle delay for synchronization is incurred before the interrupt can affect the processor flow.

Irrespective of whether the exception was entered from ARM or Thumb state, a FIQ handler should leave the interrupt by executing

```
SUBS PC,R14_fiq,#4
```

FIQ may be disabled by setting the CPSR's F flag (but note that this is not possible from User mode). If the F flag is clear, ARM7TDMI checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction.

IRQ

The IRQ (Interrupt Request) exception is a normal interrupt caused by a LOW level on the **nIRQ** input. IRQ has a lower priority than FIQ and is masked out when a FIQ sequence is entered. It may be disabled at any time by setting the I bit in the CPSR, though this can only be done from a privileged (non-User) mode.

Irrespective of whether the exception was entered from ARM or Thumb state, an IRQ handler should return from the interrupt by executing

```
SUBS PC,R14_irq,#4
```

Abort

An abort indicates that the current memory access cannot be completed. It can be signalled by the external **ABORT** input. ARM7TDMI checks for the abort exception during memory access cycles.

There are two types of abort:

- *Prefetch abort:* occurs during an instruction prefetch.
- *Data abort:* occurs during a data access.

If a prefetch abort occurs, the prefetched instruction is marked as invalid, but the exception will not be taken until the instruction reaches the head of the pipeline. If the instruction is not executed - for example because a branch occurs while it is in the pipeline - the abort does not take place.

If a data abort occurs, the action taken depends on the instruction type:

- Single data transfer instructions (LDR, STR) write back modified base registers: the Abort handler must be aware of this.
- The swap instruction (SWP) is aborted as though it had not been executed.
- Block data transfer instructions (LDM, STM) complete. If write-back is set, the base is updated. If the instruction would have overwritten the base with data (ie it has the base in the transfer list), the overwriting is prevented. All register overwriting is prevented after an abort is indicated, which means in particular that R15 (always the last register to be transferred) is preserved in an aborted LDM instruction.

The abort mechanism allows the implementation of a demand paged virtual memory system. In such a system the processor is allowed to generate arbitrary addresses. When the data at an address is unavailable, the Memory Management Unit (MMU) signals an abort. The abort handler must then work out the cause of the abort, make the requested data available, and retry the aborted instruction. The application program needs no knowledge of the amount of memory available to it, nor is its state in any way affected by the abort.

After fixing the reason for the abort, the handler should execute the following irrespective of the state (ARM or Thumb):

```
SUBS PC,R14_abt,#4      ;  for a prefetch abort, or
SUBS PC,R14_abt,#8      ;  for a data abort
```

This restores both the PC and the CPSR, and retries the aborted instruction.

Software Interrupt

The software interrupt instruction (SWI) is used for entering Supervisor mode, usually to request a particular supervisor function. A SWI handler should return by executing the following irrespective of the state (ARM or Thumb):

MOV PC,R14_svc

This restores the PC and CPSR, and returns to the instruction following the SWI.

NOTE

nFIQ, nIRQ, ISYNC, LOCK, BIGEND, and ABORT pins exist only in the ARM7TDMI CPU core.

Undefined Instruction

When ARM7TDMI comes across an instruction which it cannot handle, it takes the undefined instruction trap. This mechanism may be used to extend either the THUMB or ARM instruction set by software emulation.

After emulating the failed instruction, the trap handler should execute the following irrespective of the state (ARM or Thumb):

MOVS PC,R14_und

This restores the CPSR and returns to the instruction following the undefined instruction.

Exception Vectors

The following table shows the exception vector addresses.

Table 2-3. Exception Vectors

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Exception Priorities

When multiple exceptions arise at the same time, a fixed priority system determines the order in which they are handled:

Highest priority:

1. Reset
2. Data abort
3. FIQ
4. IRQ
5. Prefetch abort

Lowest priority:

6. Undefined Instruction, Software interrupt.

Not All Exceptions Can Occur at Once:

Undefined Instruction and Software Interrupt are mutually exclusive, since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (ie the CPSR's F flag is clear), ARM7TDMI enters the data abort handler and then immediately proceeds to the FIQ vector. A normal return from FIQ will cause the data abort handler to resume execution. Placing data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection. The time for this exception entry should be added to worst-case FIQ latency calculations.

INTERRUPT LATENCIES

The worst case latency for FIQ, assuming that it is enabled, consists of the longest time the request can take to pass through the synchroniser (*Tsyncmax* if asynchronous), plus the time for the longest instruction to complete (*Tldm*, the longest instruction is an LDM which loads all the registers including the PC), plus the time for the data abort entry (*Texc*), plus the time for FIQ entry (*Tfiq*). At the end of this time ARM7TDMI will be executing the instruction at 0x1C.

Tsyncmax is 3 processor cycles, *Tldm* is 20 cycles, *Texc* is 3 cycles, and *Tfiq* is 2 cycles. The total time is therefore 28 processor cycles. This is just over 1.4 microseconds in a system which uses a continuous 20 MHz processor clock. The maximum IRQ latency calculation is similar, but must allow for the fact that FIQ has higher priority and could delay entry into the IRQ handling routine for an arbitrary length of time. The minimum latency for FIQ or IRQ consists of the shortest time the request can take through the synchroniser (*Tsyncmin*) plus *Tfiq*. This is 4 processor cycles.

RESET

When the **nRESET** signal goes LOW, ARM7TDMI abandons the executing instruction and then continues to fetch instructions from incrementing word addresses.

When **nRESET** goes HIGH again, ARM7TDMI:

1. Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and SPSR is not defined.
2. Forces M[4:0] to 10011 (Supervisor mode), sets the I and F bits in the CPSR, and clears the CPSR's T bit.
3. Forces the PC to fetch the next instruction from address 0x00.
4. Execution resumes in ARM state.

3 Instruction set

INSTRUCTION SET SUMMAY

This chapter describes the ARM instruction set and the THUMB instruction set in the ARM7TDMI core.

FORMAT SUMMARY

The ARM instruction set formats are shown below.

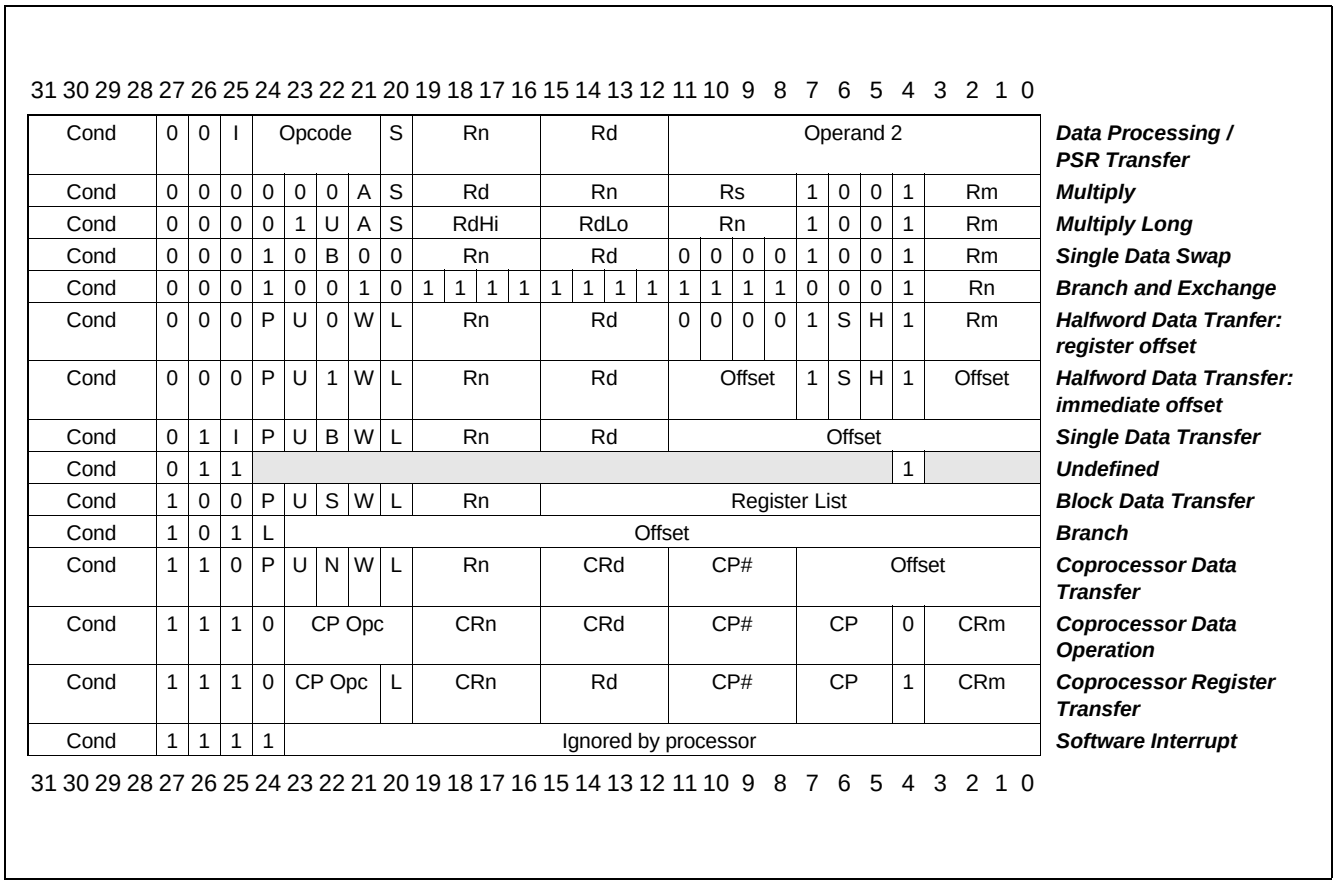

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Cond 0 0 I Opcode S Rn Rd Operand 2 Data Processing / PSR Transfer
Cond 0 0 0 0 0 0 A S Rd Rn Rs 1 0 0 1 Rm Multiply
Cond 0 0 0 0 1 U A S RdHi RdLo Rn 1 0 0 1 Rm Multiply Long
Cond 0 0 0 1 0 B 0 0 Rn Rd 0 0 0 0 1 0 0 1 Rm Single Data Swap
Cond 0 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 Rn Branch and Exchange
Cond 0 0 0 P U 0 W L Rn Rd 0 0 0 0 1 S H 1 Rm Halfword Data Tranfer: register offset
Cond 0 0 0 P U 1 W L Rn Rd Offset 1 S H 1 Offset Halfword Data Transfer: immediate offset
Cond 0 1 I P U B W L Rn Rd Offset Single Data Transfer
Cond 0 1 1 1 Undefined
Cond 1 0 0 P U S W L Rn Register List Block Data Transfer
Cond 1 0 1 L Offset Branch
Cond 1 1 0 P U N W L Rn CRd CP# Offset Coprocessor Data Transfer
Cond 1 1 1 0 CP Opc CRn CRd CP# CP 0 CRm Coprocessor Data Operation
Cond 1 1 1 0 CP Opc L CRn Rd CP# CP 1 CRm Coprocessor Register Transfer
Cond 1 1 1 1 Ignored by processor Software Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Figure 3-1. ARM Instruction Set Format

NOTE

Some instruction codes are not defined but do not cause the Undefined instruction trap to be taken, for instance a Multiply instruction with bit 6 changed to a 1. These instructions should not be used, as their action may change in future ARM implementations.

INSTRUCTION SUMMARY

Table 3-1. The ARM Instruction Set

Mnemonic	Instruction	Action
ADC	Add with carry	Rd: = Rn + Op2 + Carry
ADD	Add	Rd: = Rn + Op2
AND	AND	Rd: = Rn AND Op2
B	Branch	R15: = address
BIC	Bit Clear	Rd: = Rn AND NOT Op2
BL	Branch with Link	R14: = R15, R15: = address
BX	Branch and Exchange	R15: = Rn, T bit: = Rn[0]
CDP	Coprocesor Data Processing	(Coprocessor-specific)
CMN	Compare Negative	CPSR flags: = Rn + Op2
CMP	Compare	CPSR flags: = Rn – Op2
EOR	Exclusive OR	Rd: = (Rn AND NOT Op2) OR (op2 AND NOT Rn)
LDC	Load coprocessor from memory	Coprocessor load
LDM	Load multiple registers	Stack manipulation (Pop)
LDR	Load register from memory	Rd: = (address)
MCR	Move CPU register to coprocessor register	cRn: = rRn {<op>cRm}
MLA	Multiply Accumulate	Rd: = (Rm * Rs) + Rn
MOV	Move register or constant	Rd: = Op2
MRC	Move from coprocessor register to CPU register	Rn: = cRn {<op>cRm}
MRS	Move PSR status/flags to register	Rn: = PSR
MSR	Move register to PSR status/flags	PSR: = Rm
MUL	Multiply	Rd: = Rm * Rs
MVN	Move negative register	Rd: = 0xFFFFFFFF EOR Op2
ORR	OR	Rd: = Rn OR Op2
RSB	Reverse Subtract	Rd: = Op2 – Rn
RSC	Reverse Subtract with Carry	Rd: = Op2 – Rn – 1 + Carry
SBC	Subtract with Carry	Rd: = Rn – Op2 - 1 + Carry
STC	Store coprocessor register to memory	address: = CRn
STM	Store Multiple	Stack manipulation (Push)
STR	Store register to memory	<address>: = Rd
SUB	Subtract	Rd: = Rn – Op2
SWI	Software Interrupt	OS call
SWP	Swap register with memory	Rd: = [Rn], [Rn] := Rm
TEQ	Test bitwise equality	CPSR flags: = Rn EOR Op2
TST	Test bits	CPSR flags: = Rn AND Op2

THE CONDITION FIELD

In ARM state, all instructions are conditionally executed according to the state of the CPSR condition codes and the instruction's condition field. This field (bits 31:28) determines the circumstances under which an instruction is to be executed. If the state of the C, N, Z and V flags fulfils the conditions encoded by the field, the instruction is executed, otherwise it is ignored.

There are sixteen possible conditions, each represented by a two-character suffix that can be appended to the instruction's mnemonic. For example, a Branch (B in assembly language) becomes BEQ for "Branch if Equal", which means the Branch will only be taken if the Z flag is set.

In practice, fifteen different conditions may be used: these are listed in Table 3-2. The sixteenth (1111) is reserved, and must not be used.

In the absence of a suffix, the condition field of most instructions is set to "Always" (sufix AL). This means the instruction will always be executed regardless of the CPSR condition codes.

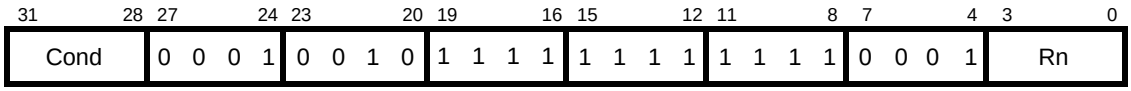
Table 3-2. Condition Code Summary

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

BRANCH AND EXCHANGE (BX)

This instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.

This instruction performs a branch by copying the contents of a general register, Rn, into the program counter, PC. The branch causes a pipeline flush and refill from the address specified by Rn. This instruction also permits the instruction set to be exchanged. When the instruction is executed, the value of Rn[0] determines whether the instruction stream will be decoded as ARM or THUMB instructions.



[3:0] Operand register

If bit 0 of Rn = 1, subsequent instructions decoded as THUMB instructions
If bit 0 of Rn = 0, subsequent instructions decoded as ARM instructions

[31:28] Condition Field

Figure 3-2. Branch and Exchange Instructions

INSTRUCTION CYCLE TIMES

The BX instruction takes 2S + 1N cycles to execute, where S and N are defined as sequential (S-cycle) and non-sequencial (N-cycle), respectively.

ASSEMBLER SYNTAX

BX - branch and exchange.

BX {cond} Rn

{cond} Two character condition mnemonic. See Table 3-2.

Rn is an expression evaluating to a valid register number.

USING R15 AS AN OPERAND

If R15 is used as an operand, the behaviour is undefined.

Examples

```
ADR       R0, Into_THUMB + 1    ;  Generate branch target address
                                ;  and set bit 0 high - hence
                                ;  arrive in THUMB state.
BX        R0                    ;  Branch and change to THUMB
                                ;  state.
CODE16                          ;  Assemble subsequent code as
Into_THUMB                      ;  THUMB instructions
.
.
.
ADR R5, Back_to_ARM             :  Generate branch target to word aligned address
                                ;  - hence bit 0 is low and so change back to ARM state.
BX R5                           ;  Branch and change back to ARM state.
.
.
.
ALIGN                           ;  Word align
CODE32                          ;  Assemble subsequent code as ARM instructions
Back_to_ARM
```

BRANCH AND BRANCH WITH LINK (B, BL)

The instruction is only executed if the condition is true. The various conditions are defined Table 3-2. The instruction encoding is shown in Figure 3-3, below.

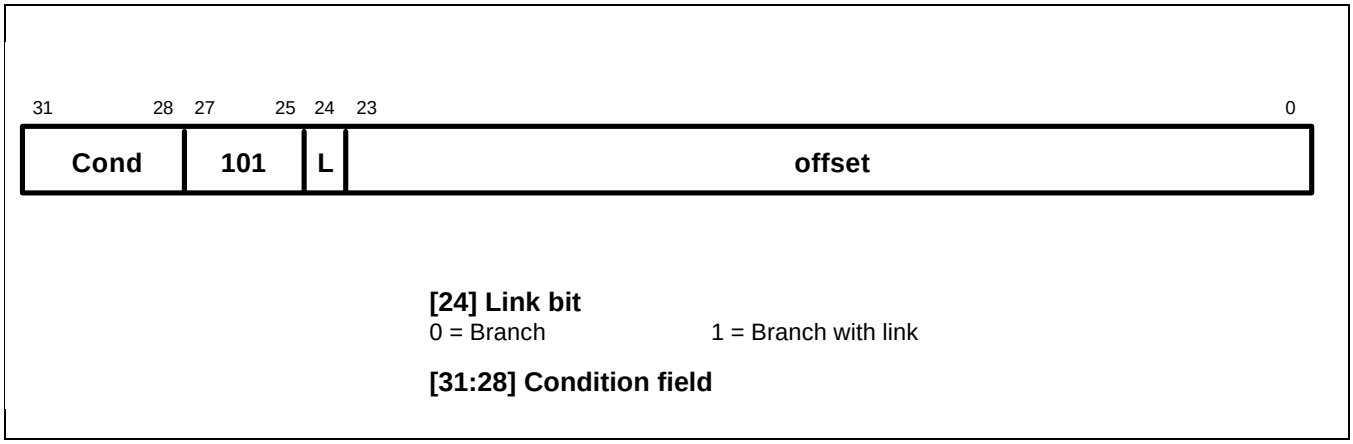


Figure 3-3. Branch Instructions

Branch instructions contain a signed 2's complement 24 bit offset. This is shifted left two bits, sign extended to 32 bits, and added to the PC. The instruction can therefore specify a branch of +/- 32Mbytes. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words (8 bytes) ahead of the current instruction.

Branches beyond +/- 32Mbytes must use an offset or absolute destination which has been previously loaded into a register. In this case the PC should be manually saved in R14 if a Branch with Link type operation is required.

THE LINK BIT

Branch with Link (BL) writes the old PC into the link register (R14) of the current bank. The PC value written into R14 is adjusted to allow for the prefetch, and contains the address of the instruction following the branch and link instruction. Note that the CPSR is not saved with the PC and R14[1:0] are always cleared.

To return from a routine called by Branch with Link use MOV PC,R14 if the link register is still valid or LDM Rn!, {..PC} if the link register has been saved onto a stack pointed to by Rn.

INSTRUCTION CYCLE TIMES

Branch and Branch with Link instructions take 2S + 1N incremental cycles, where S and N are defined as squential (S-cycle) and internal (I-cycle).

ASSEMBLER SYNTAX

Items in {} are optional. Items in <> must be present.

B{L}{cond} <expression>

{L}	Used to request the Branch with Link form of the instruction. If absent, R14 will not be affected by the instruction.
{cond}	A two-character mnemonic as shown in Table 3-2. If absent then AL (ALways) will be used.
<expression>	The destination. The assembler calculates the offset.

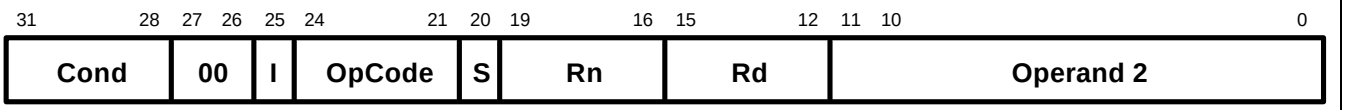
EXAMPLES

```
here    BAL     here        ;  Assembles to 0xEAFFFFFE (note effect of PC offset).
        B       there       ;  Always condition used as default.
        CMP     R1,#0       ;  Compare R1 with zero and branch to fred
                            ;  if R1 was zero, otherwise continue.
        BEQ     fred        ;  Continue to next instruction.

        BL      sub+ROM     ;  Call subroutine at computed address.
        ADDS    R1,#1       ;  Add 1 to register 1, setting CPSR flags
                            ;  on the result then call subroutine if
        BLCC    sub         ;  the C flag is clear, which will be the
                            ;  case unless R1 held 0xFFFFFFFF.
```

DATA PROCESSING

The data processing instruction is only executed if the condition is true. The conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-4.



[15:12] Destination register
0 = Branch 1 = Branch with Link

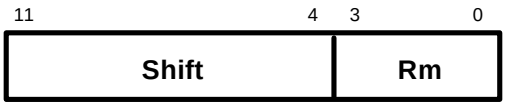
[19:16] 1st operand register
0 = Branch 1 = Branch with Link

[20] Set condition codes
0 = Do not after condition codes 1 = Set condition codes

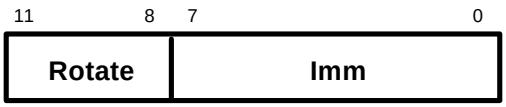
[24:21] Operation code
0000 = AND - Rd: = Op1 AND Op2
0001 = EOR - Rd: = Op1 EOR Op2
0010 = SUB - Rd: = Op1 - Op2
0011 = RSB - Rd: = Op2 - Op1
0100 = ADD - Rd: = Op1 + Op2
0101 = ADC - Rd: = Op1 + Op2 + C
0110 = SBC - Rd: = Op1 - Op2 + C - 1
0111 = RSC - Rd: = Op2 - Op1 + C - 1
1000 = TST - set condition codes on Op 1 AND Op2
1001 = TEO - set condition codes on Op1 EOR Op2
1010 = CMP - set condition codes on Op1 - Op2
1011 = SMN - set condition codes on Op1 + Op2
1100 = ORR - Rd: = Op1 OR Op2
1101 = MOV - Rd: = Op2
1110 = BIC - Rd: = Op1 AND NOT Op2
1111 = MVN - Rd: = NOT Op2

[25] Immediate operand
0 = Operand 2 is a register 1 = Operand 2 is an Immediate Value

[11:0] Operand 2 type selection



[3:0] 2nd Operand Register [11:4] Shift Applied to Rm



[7:0] Unsigned 8 bit immediate value [11:8] Shift applied to Imm

[31:28] Condition field

Figure 3-4. Data Processing Instructions

The instruction produces a result by performing a specified arithmetic or logical operation on one or two operands. The first operand is always a register (Rn).

The second operand may be a shifted register (Rm) or a rotated 8 bit immediate value (Imm) according to the value of the I bit in the instruction. The condition codes in the CPSR may be preserved or updated as a result of this instruction, according to the value of the S bit in the instruction.

Certain operations (TST, TEQ, CMP, CMN) do not write the result to Rd. They are used only to perform tests and to set the condition codes on the result and always have the S bit set. The instructions and their effects are listed in Table 3-3.

CPSR FLAGS

The data processing operations may be classified as logical or arithmetic. The logical operations (AND, EOR, TST, TEQ, ORR, MOV, BIC, MVN) perform the logical action on all corresponding bits of the operand or operands to produce the result. If the S bit is set (and Rd is not R15, see below) the V flag in the CPSR will be unaffected, the C flag will be set to the carry out from the barrel shifter (or preserved when the shift operation is LSL #0), the Z flag will be set if and only if the result is all zeros, and the N flag will be set to the logical value of bit 31 of the result .

Table 3-3. ARM Data Processing Instructions

Assembler Mnemonic	Op-Code	Action
AND	0000	Operand1 AND operand2
EOR	0001	Operand1 EOR operand2
SUB	0010	Operand1 – operand2
RSB	0011	Operand2 operand1
ADD	0100	Operand1 + operand2
ADC	0101	Operand1 + operand2 + carry
SBC	0110	Operand1 – operand2 + carry – 1
RSC	0111	Operand2 – operand1 + carry – 1
TST	1000	As AND, but result is not written
TEQ	1001	As EOR, but result is not written
CMP	1010	As SUB, but result is not written
CMN	1011	As ADD, but result is not written
ORR	1100	Operand1 OR operand2
MOV	1101	Operand2 (operand1 is ignored)
BIC	1110	Operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

The arithmetic operations (SUB, RSB, ADD, ADC, SBC, RSC, CMP, CMN) treat each operand as a 32 bit integer (either unsigned or 2's complement signed, the two are equivalent). If the S bit is set (and Rd is not R15) the V flag in the CPSR will be set if an overflow occurs into bit 31 of the result; this may be ignored if the operands were considered unsigned, but warns of a possible error if the operands were 2's complement signed. The C flag will be set to the carry out of bit 31 of the ALU, the Z flag will be set if and only if the result was zero, and the N flag will be set to the value of bit 31 of the result (indicating a negative result if the operands are considered to be 2's complement signed).

SHIFTS

When the second operand is specified to be a shifted register, the operation of the barrel shifter is controlled by the Shift field in the instruction. This field indicates the type of shift to be performed (logical left or right, arithmetic right or rotate right). The amount by which the register should be shifted may be contained in an immediate field in the instruction, or in the bottom byte of another register (other than R15). The encoding for the different shift types is shown in Figure 3-5.

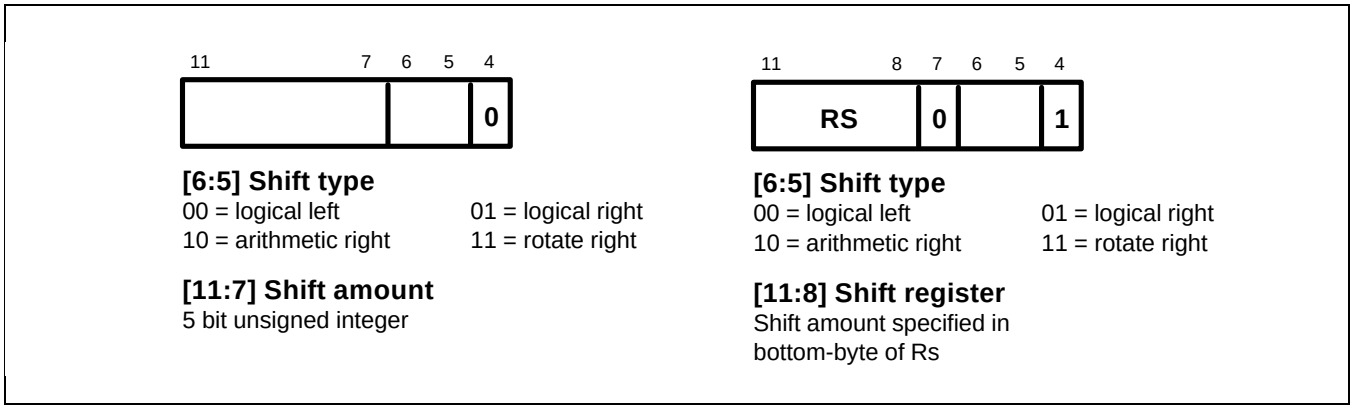


Figure 3-5. ARM Shift Operations

Instruction specified shift amount

When the shift amount is specified in the instruction, it is contained in a 5 bit field which may take any value from 0 to 31. A logical shift left (LSL) takes the contents of Rm and moves each bit by the specified amount to a more significant position. The least significant bits of the result are filled with zeros, and the high bits of Rm which do not map into the result are discarded, except that the least significant discarded bit becomes the shifter carry output which may be latched into the C bit of the CPSR when the ALU operation is in the logical class (see above). For example, the effect of LSL #5 is shown in Figure 3-6.

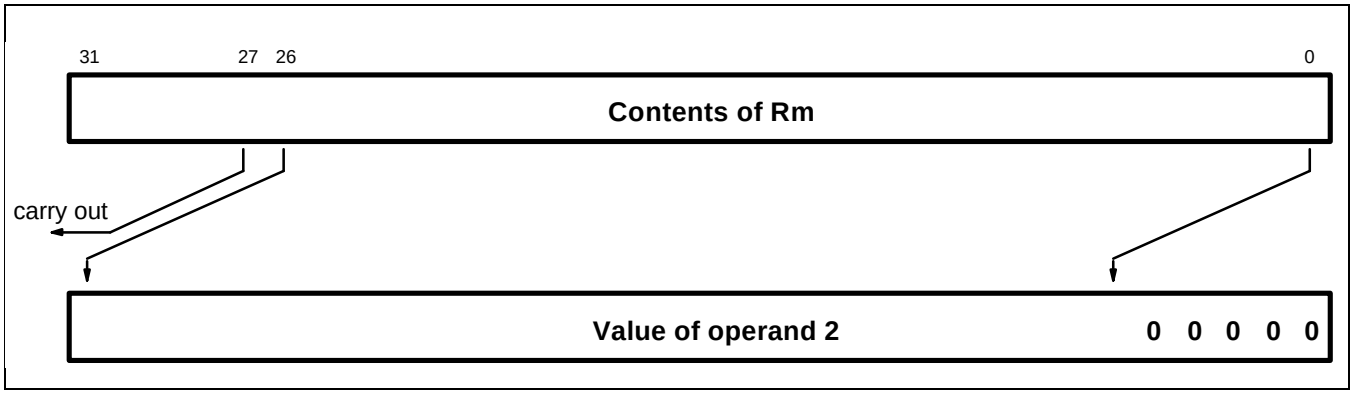


Figure 3-6. Logical Shift Left

NOTE

LSL #0 is a special case, where the shifter carry out is the old value of the CPSR C flag. The contents of Rm are used directly as the second operand. A logical shift right (LSR) is similar, but the contents of Rm are moved to less significant positions in the result. LSR #5 has the effect shown in Figure 3-7.

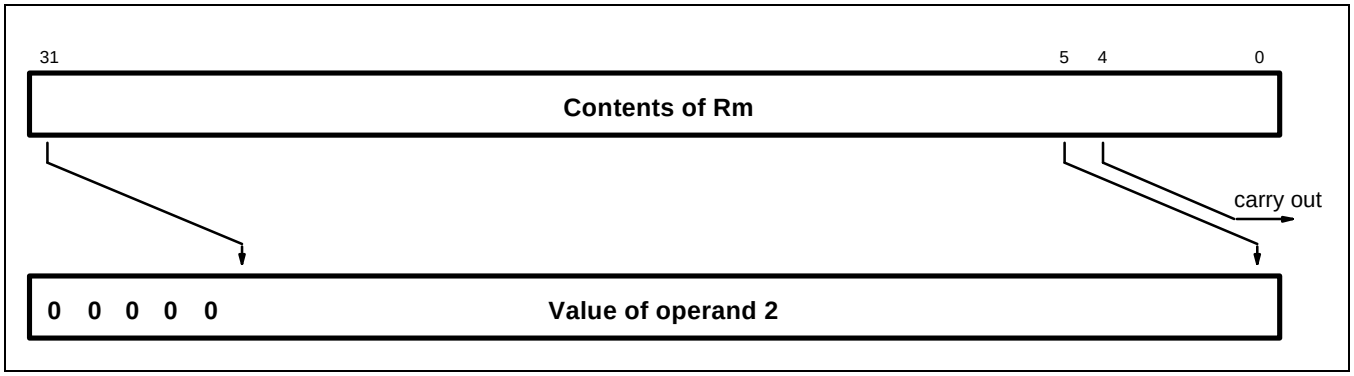


Figure 3-7. Logical Shift Right

The form of the shift field which might be expected to correspond to LSR #0 is used to encode LSR #32, which has a zero result with bit 31 of Rm as the carry output. Logical shift right zero is redundant as it is the same as logical shift left zero, so the assembler will convert LSR #0 (and ASR #0 and ROR #0) into LSL #0, and allow LSR #32 to be specified.

An arithmetic shift right (ASR) is similar to logical shift right, except that the high bits are filled with bit 31 of Rm instead of zeros. This preserves the sign in 2's complement notation. For example, ASR #5 is shown in Figure 3-8.

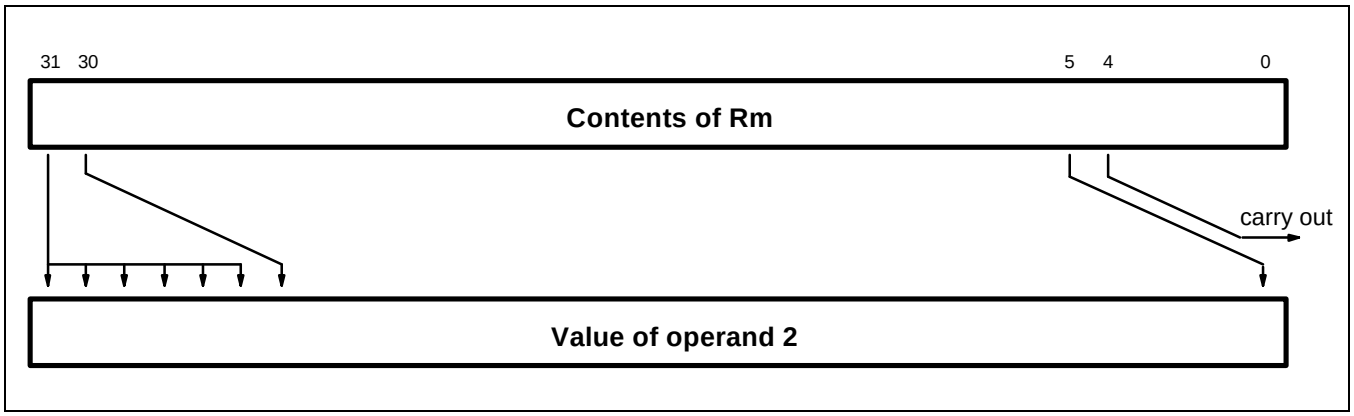


Figure 3-8. Arithmetic Shift Right

The form of the shift field which might be expected to give ASR #0 is used to encode ASR #32. Bit 31 of Rm is again used as the carry output, and each bit of operand 2 is also equal to bit 31 of Rm. The result is therefore all ones or all zeros, according to the value of bit 31 of Rm.

Rotate right (ROR) operations reuse the bits which “overshoot” in a logical shift right operation by reintroducing them at the high end of the result, in place of the zeros used to fill the high end in logical right operations. For example, ROR #5 is shown in Figure 3-9.

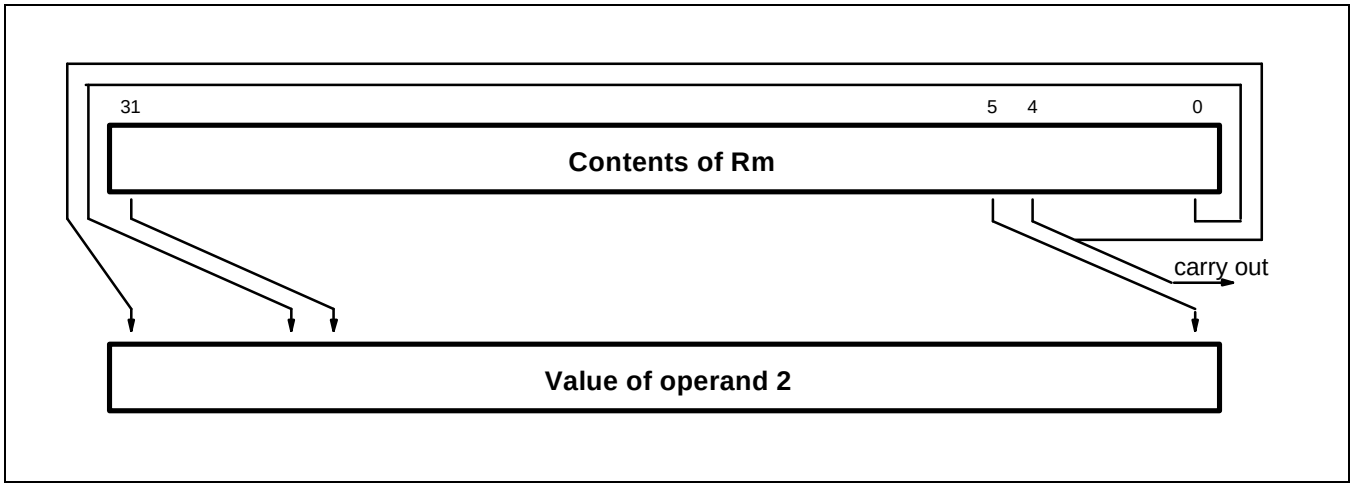


Figure 3-9. Rotate Right

The form of the shift field which might be expected to give ROR #0 is used to encode a special function of the barrel shifter, rotate right extended (RRX). This is a rotate right by one bit position of the 33 bit quantity formed by appending the CPSR C flag to the most significant end of the contents of Rm as shown in Figure 3-10.

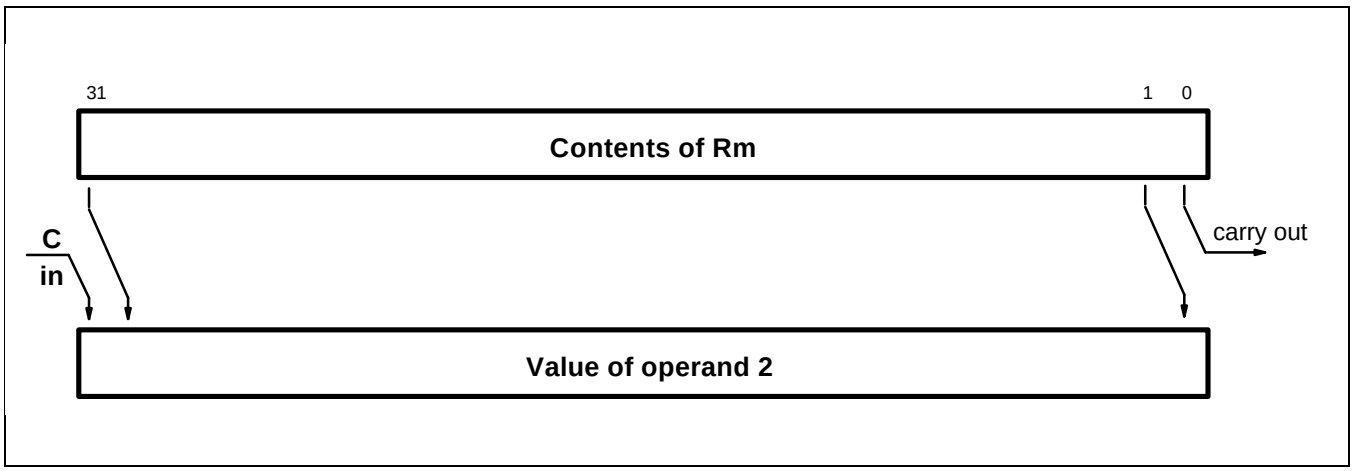


Figure 3-10. Rotate Right Extended

Register specified shift amount

Only the least significant byte of the contents of Rs is used to determine the shift amount. Rs can be any general register other than R15.

If this byte is zero, the unchanged contents of Rm will be used as the second operand, and the old value of the CPSR C flag will be passed on as the shifter carry output.

If the byte has a value between 1 and 31, the shifted result will exactly match that of an instruction specified shift with the same value and shift operation.

If the value in the byte is 32 or more, the result will be a logical extension of the shift described above:

1 LSL by 32 has result zero, carry out equal to bit 0 of Rm.

2 LSL by more than 32 has result zero, carry out zero.

3 LSR by 32 has result zero, carry out equal to bit 31 of Rm.

4 LSR by more than 32 has result zero, carry out zero.

5 ASR by 32 or more has result filled with and carry out equal to bit 31 of Rm.

6 ROR by 32 has result equal to Rm, carry out equal to bit 31 of Rm.

7 ROR by n where n is greater than 32 will give the same result and carry out as ROR by n-32; therefore repeatedly subtract 32 from n until the amount is in the range 1 to 32 and see above.

NOTE

The zero in bit 7 of an instruction with a register controlled shift is compulsory; a one in this bit will cause the instruction to be a multiply or undefined instruction.

IMMEDIATE OPERAND ROTATES

The immediate operand rotate field is a 4 bit unsigned integer which specifies a shift operation on the 8 bit immediate value. This value is zero extended to 32 bits, and then subject to a rotate right by twice the value in the rotate field. This enables many common constants to be generated, for example all powers of 2.

WRITING TO R15

When Rd is a register other than R15, the condition code flags in the CPSR may be updated from the ALU flags as described above.

When Rd is R15 and the S flag in the instruction is not set the result of the operation is placed in R15 and the CPSR is unaffected.

When Rd is R15 and the S flag is set the result of the operation is placed in R15 and the SPSR corresponding to the current mode is moved to the CPSR. This allows state changes which atomically restore both PC and CPSR. This form of instruction should not be used in User mode.

USING R15 AS AN OPERAND

If R15 (the PC) is used as an operand in a data processing instruction the register is used directly.

The PC value will be the address of the instruction, plus 8 or 12 bytes due to instruction prefetching. If the shift amount is specified in the instruction, the PC will be 8 bytes ahead. If a register is used to specify the shift amount the PC will be 12 bytes ahead.

TEQ, TST, CMP AND CMN OPCODES

NOTE

TEQ, TST, CMP and CMN do not write the result of their operation but do set flags in the CPSR. An assembler should always set the S flag for these instructions even if this is not specified in the mnemonic.

The TEQP form of the TEQ instruction used in earlier ARM processors must not be used: the PSR transfer operations should be used instead.

The action of TEQP in the ARM7TDMI is to move SPSR_<mode> to the CPSR if the processor is in a privileged mode and to do nothing if in User mode.

INSTRUCTION CYCLE TIMES

Data Processing instructions vary in the number of incremental cycles taken as follows:

Table 3-4. Incremental Cycle Times

Processing Type	Cycles
Normal Data Processing	1S
Data Processing with register specified shift	1S + 1I
Data Processing with PC written	2S + 1N
Data Processing with register specified shift and PC written	2S + 1N + 1I

NOTE: S, N and I are as defined sequential (S-cycle), non-sequencial (N-cycle), and internal (I-cycle) respectively .

ASSEMBLER SYNTAX

- MOV,MVN (single operand instructions).
  <opcode>{cond}{S} Rd,<Op2>
- CMP,CMN,TEQ,TST (instructions which do not produce a result).
  <opcode>{cond} Rn,<Op2>
- AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC
  <opcode>{cond}{S} Rd,Rn,<Op2>

where:

<Op2>	Rm{,<shift>} or,<#expression>
{cond}	A two-character condition mnemonic. See Table 3-2.
{S}	Set condition codes if S present (implied for CMP, CMN, TEQ, TST).
Rd, Rn and Rm	Expressions evaluating to a register number.
<#expression>	If this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.
<shift>	<Shiftname> <register> or <shiftname> #expression, or RRX (rotate right one bit with extend).
<shiftname>s	ASL, LSL, LSR, ASR, ROR. (ASL is a synonym for LSL, they assemble to the same code.)

EXAMPLES

```
ADDEQ   R2,R4,R5        ; If the Z flag is set make R2:=R4+R5
TEQS    R4,#3           ; Test R4 for equality with 3.
                        ; (The S is in fact redundant as the
                        ; assembler inserts it automatically.)
SUB     R4,R5,R7,LSR R2 ; Logical right shift R7 by the number in
                        ; the bottom byte of R2, subtract result
                        ; from R5, and put the answer into R4.
MOV     PC,R14          ; Return from subroutine.
MOVS    PC,R14          ; Return from exception and restore CPSR
                        ; from SPSR_mode.
```

PSR TRANSFER (MRS, MSR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.

The MRS and MSR instructions are formed from a subset of the Data Processing operations and are implemented using the TEQ, TST, CMN and CMP instructions without the S flag set. The encoding is shown in Figure 3-11.

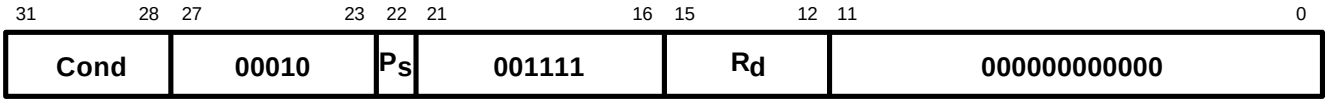
These instructions allow access to the CPSR and SPSR registers. The MRS instruction allows the contents of the CPSR or SPSR_<mode> to be moved to a general register. The MSR instruction allows the contents of a general register to be moved to the CPSR or SPSR_<mode> register.

The MSR instruction also allows an immediate value or register contents to be transferred to the condition code flags (N,Z,C and V) of CPSR or SPSR_<mode> without affecting the control bits. In this case, the top four bits of the specified register contents or 32 bit immediate value are written to the top four bits of the relevant PSR.

OPERAND RESTRICTIONS

- In user mode, the control bits of the CPSR are protected from change, so only the condition code flags of the CPSR can be changed. In other (privileged) modes the entire CPSR can be changed.
- Note that the software must never change the state of the T bit in the CPSR. If this happens, the processor will enter an unpredictable state.
- The SPSR register which is accessed depends on the mode at the time of execution. For example, only SPSR_fiq is accessible when the processor is in FIQ mode.
- You must not specify R15 as the source or destination register.
- Also, do not attempt to access an SPSR in User mode, since no such register exists.

MRS (transfer PSR contents to a register)

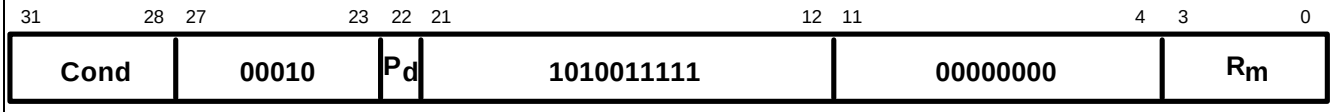


[15:12] Destination register

[22] Source PSR
0 = CPSR 1 = SPSR_<current mode>

[31:28] Condition field

MRS (transfer register contents to PSR)

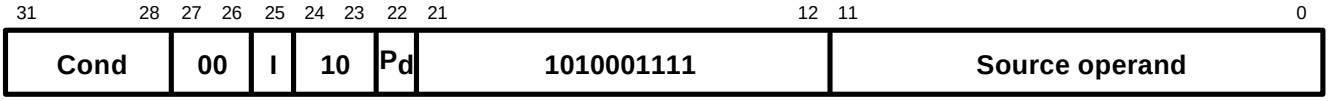


[3:0] Source register

[22] Destination PSR
0 = CPSR 1 = SPSR_<current mode>

[31:28] Condition field

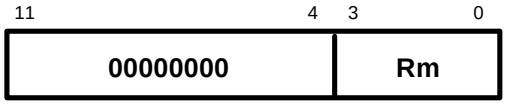
MRS (transfer register contents or immdiate value to PSR flag bits only)



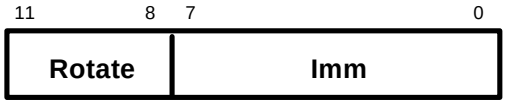
[22] Destination PSR
0 = CPSR 1 = SPSR_<current mode>

[25] Immediate Operand
0 = Source operand is a register
1 = SPSR_<current mode>

[11:0] Source operand



[3:0] Source register
[11:4] Source operand is an immediate value



[7:0] Unsigned 8 bit immediate value
[11:8] Shift applied to Imm

[31:28] Condition field

Figure 3-11. PSR Transfer

RESERVED BITS

Only twelve bits of the PSR are defined in ARM7TDMI (N,Z,C,V,I,F, T & M[4:0]); the remaining bits are reserved for use in future versions of the processor. Refer to Figure 2-6 for a full description of the PSR bits.

To ensure the maximum compatibility between ARM7TDMI programs and future processors, the following rules should be observed:

- The reserved bits should be preserved when changing the value in a PSR.
- Programs should not rely on specific values from the reserved bits when checking the PSR status, since they may read as one or zero in future processors.

A read-modify-write strategy should therefore be used when altering the control bits of any PSR register; this involves transferring the appropriate PSR register to a general register using the MRS instruction, changing only the relevant bits and then transferring the modified value back to the PSR register using the MSR instruction.

EXAMPLES

The following sequence performs a mode change:

```
MRS     R0,CPSR          ;  Take a copy of the CPSR.
BIC     R0,R0,#0x1F      ;  Clear the mode bits.
ORR     R0,R0,#new_mode  ;  Select new mode
MSR     CPSR,R0          ;  Write back the modified CPSR.
```

When the aim is simply to change the condition code flags in a PSR, a value can be written directly to the flag bits without disturbing the control bits. The following instruction sets the N,Z,C and V flags:

```
MSR     CPSR_flg,#0xF0000000   ;  Set all the flags egardless of their previous state
                               ;  (does not affect any control bits).
```

No attempt should be made to write an 8 bit immediate value into the whole PSR since such an operation cannot preserve the reserved bits.

INSTRUCTION CYCLE TIMES

PSR transfers take 1S incremental cycles, where S is defined as Sequential (S-cycle).

ASSEMBLER SYNTAX

- MRS - transfer PSR contents to a register
  MRS{cond} Rd,<psr>
- MSR - transfer register contents to PSR
  MSR{cond} <psr>,Rm
- MSR - transfer register contents to PSR flag bits only
  MSR{cond} <psrf>,Rm

The most significant four bits of the register contents are written to the N,Z,C & V flags respectively.

- MSR - transfer immediate value to PSR flag bits only
  MSR{cond} <psrf>,<#expression>

The expression should symbolise a 32 bit value of which the most significant four bits are written to the N,Z,C and V flags respectively.

Key:

{cond}	Two-character condition mnemonic. See Table 3-2..
Rd and Rm	Expressions evaluating to a register number other than R15
<psr>	CPSR, CPSR_all, SPSR or SPSR_all. (CPSR and CPSR_all are synonyms as are SPSR and SPSR_all)
<psrf>	CPSR_flg or SPSR_flg
<#expression>	Where this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.

EXAMPLES

In User mode the instructions behave as follows:

```
MSR     CPSR_all,Rm             ;  CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg,Rm             ;  CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg,#0xA0000000    ;  CPSR[31:28] <- 0xA (set N,C; clear Z,V)
MRS     Rd,CPSR                 ;  Rd[31:0] <- CPSR[31:0]
```

In privileged modes the instructions behave as follows:

```
MSR     CPSR_all,Rm             ;  CPSR[31:0]  <- Rm[31:0]
MSR     CPSR_flg,Rm             ;  CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg,#0x50000000    ;  CPSR[31:28] <- 0x5 (set Z,V; clear N,C)
MSR     SPSR_all,Rm             ;  SPSR_<mode>[31:0]<- Rm[31:0]
MSR     SPSR_flg,Rm             ;  SPSR_<mode>[31:28] <- Rm[31:28]
MSR     SPSR_flg,#0xC0000000    ;  SPSR_<mode>[31:28] <- 0xC (set N,Z; clear C,V)
MRS     Rd,SPSR                 ;  Rd[31:0] <- SPSR_<mode>[31:0]
```

MULTIPLY AND MULTIPLY-ACCUMULATE (MUL, MLA)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-12.

The multiply and multiply-accumulate instructions use an 8 bit Booth's algorithm to perform integer multiplication.

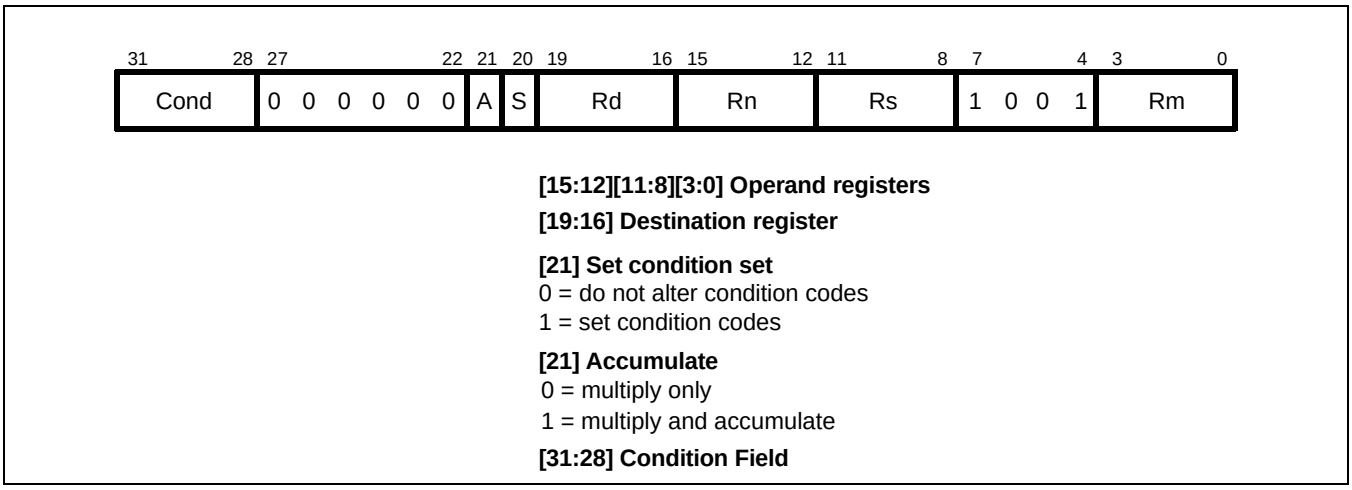


Figure 3-12. Multiply Instructions

The multiply form of the instruction gives Rd:=Rm*Rs. Rn is ignored, and should be set to zero for compatibility with possible future upgrades to the instruction set.The multiply-accumulate form gives Rd:=Rm*Rs+Rn, which can save an explicit ADD instruction in some circumstances. Both forms of the instruction work on operands which may be considered as signed (2's complement) or unsigned integers.

The results of a signed multiply and of an unsigned multiply of 32 bit operands differ only in the upper 32 bits—the low 32 bits of the signed and unsigned results are identical. As these instructions only produce the low 32 bits of a multiply, they can be used for both signed and unsigned multiplies.

For example consider the multiplication of the operands:

Operand A	Operand B	Result
0xFFFFFFF6	0x0000014	0xFFFFFF38

If the Operands Are Interpreted as Signed

Operand A has the value -10, operand B has the value 20, and the result is -200 which is correctly represented as 0xFFFFFF38.

If the Operands Are Interpreted as Unsigned

Operand A has the value 4294967286, operand B has the value 20 and the result is 85899345720, which is represented as 0x13FFFFFF38, so the least significant 32 bits are 0xFFFFFF38.

Operand Restrictions

The destination register Rd must not be the same as the operand register Rm. R15 must not be used as an operand or as the destination register.

All other register combinations will give correct results, and Rd, Rn and Rs may use the same register when required.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N (Negative) and Z (Zero) flags are set correctly on the result (N is made equal to bit 31 of the result, and Z is set if and only if the result is zero). The C (Carry) flag is set to a meaningless value and the V (oVerflow) flag is unaffected.

INSTRUCTION CYCLE TIMES

MUL takes 1S + mI and MLA 1S + (m+1)I cycles to execute, where S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.

m	The number of 8 bit multiplier array cycles is required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs. Its possible values are as follows
1	If bits [32:8] of the multiplier operand are all zero or all one.
2	If bits [32:16] of the multiplier operand are all zero or all one.
3	If bits [32:24] of the multiplier operand are all zero or all one.
4	In all other cases.

ASSEMBLER SYNTAX

MUL{cond}{S} Rd,Rm,Rs
MLA{cond}{S} Rd,Rm,Rs,Rn

{cond}	Two-character condition mnemonic. See Table 3-2..
{S}	Set condition codes if S present
Rd, Rm, Rs and Rn	Expressions evaluating to a register number other than R15.

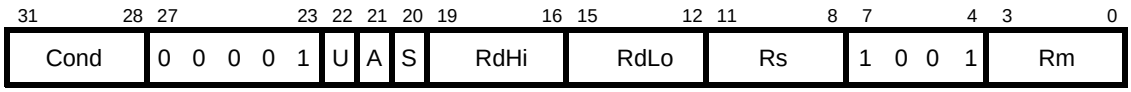
EXAMPLES

```
MUL     R1,R2,R3        ;  R1:=R2*R3
MLAEQS  R1,R2,R3,R4     ;  Conditionally R1:=R2*R3+R4, Setting condition codes.
```

MULTIPLY LONG AND MULTIPLY-ACCUMULATE LONG (MULL,MLAL)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-13.

The multiply long instructions perform integer multiplication on two 32 bit operands and produce 64 bit results. Signed and unsigned multiplication each with optional accumulate give rise to four variations.



[11:8][3:0] Operand registers
[19:16][15:12] Source destination registers

[20] Set condition code
0 = do not alter condition codes
1 = set condition codes

[21] Accumulate
0 = multiply only
1 = multiply and accumulate

[22] Unsigned
0 = unsigned
1 = signed

[31:28] Condition Field

Figure 3-13. Multiply Long Instructions

The multiply forms (UMULL and SMULL) take two 32 bit numbers and multiply them to produce a 64 bit result of the form RdHi,RdLo := Rm * Rs. The lower 32 bits of the 64 bit result are written to RdLo, the upper 32 bits of the result are written to RdHi.

The multiply-accumulate forms (UMLAL and SMLAL) take two 32 bit numbers, multiply them and add a 64 bit number to produce a 64 bit result of the form RdHi,RdLo := Rm * Rs + RdHi,RdLo. The lower 32 bits of the 64 bit number to add is read from RdLo. The upper 32 bits of the 64 bit number to add is read from RdHi. The lower 32 bits of the 64 bit result are written to RdLo. The upper 32 bits of the 64 bit result are written to RdHi.

The UMULL and UMLAL instructions treat all of their operands as unsigned binary numbers and write an unsigned 64 bit result. The SMULL and SMLAL instructions treat all of their operands as two's-complement signed numbers and write a two's-complement signed 64 bit result.

OPERAND RESTRICTIONS

- R15 must not be used as an operand or as a destination register.
- RdHi, RdLo, and Rm must all specify different registers.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N and Z flags are set correctly on the result (N is equal to bit 63 of the result, Z is set if and only if all 64 bits of the result are zero). Both the C and V flags are set to meaningless values.

INSTRUCTION CYCLE TIMES

MULL takes 1S + (m+1)I and MLAL 1S + (m+2)I cycles to execute, where *m* is the number of 8 bit multiplier array cycles required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs.

Its possible values are as follows:

For Signed Instructions SMULL, SMLAL:

- If bits [31:8] of the multiplier operand are all zero or all one.
- If bits [31:16] of the multiplier operand are all zero or all one.
- If bits [31:24] of the multiplier operand are all zero or all one.
- In all other cases.

For Unsigned Instructions UMULL, UMLAL:

- If bits [31:8] of the multiplier operand are all zero.
- If bits [31:16] of the multiplier operand are all zero.
- If bits [31:24] of the multiplier operand are all zero.
- In all other cases.

S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

Table 3-5. Assembler Syntax Descriptions

Mnemonic	Description	Purpose
UMULL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply Long	32 x 32 = 64
UMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply & Accumulate Long	32 x 32 + 64 = 64
SMULL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply Long	32 x 32 = 64
SMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply & Accumulate Long	32 x 32 + 64 = 64

where:

{cond}	Two-character condition mnemonic. See Table 3-2.
{S}	Set condition codes if S present
RdLo, RdHi, Rm, Rs	Expressions evaluating to a register number other than R15.

EXAMPLES

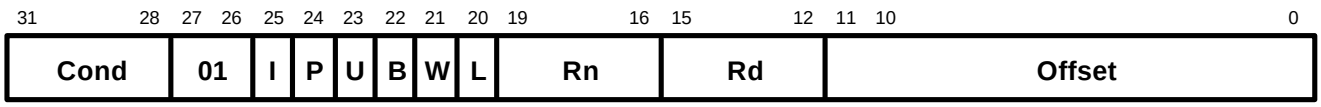
```
        UMULL   R1,R4,R2,R3     ;  R4,R1:=R2*R3
        UMLALS  R1,R5,R2,R3     ;  R5,R1:=R2*R3+R5,R1 also setting condition codes
```

SINGLE DATA TRANSFER (LDR, STR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-14.

The single data transfer instructions are used to load or store single bytes or words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register.

The result of this calculation may be written back into the base register if auto-indexing is required.



[15:12] Source/Destination register

[19:16] Base register

[20] Load/Store bit
0 = Store to memory
1 = Load from memory

[21] Write-back bit
0 = No write-back
1 = Write address into base

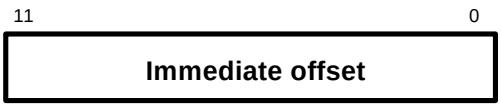
[22] Byte/Word bit
0 = Transfer word quantity
1 = Transfer byte quantity

[23] Up/Down bit
0 = Down: subtract offset from base
1 = Up: add offset to base

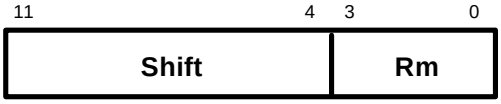
[24] Pre/Post indexing bit
0 = Post: add offset after transfer
1 = Pre: add offset before transfer

[25] Immediate offset
0 = Offset is an immediate value

[11:0] Offset



[11:0] Unsigned 12 bit immediate offset



[3:0] Offset register [11:4] Shift applied to Rm

[31:28] Condition field

Figure 3-14. Single Data Transfer Instructions

OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 12 bit unsigned binary immediate value in the instruction, or a second register (possibly shifted in some way). The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base is used as the transfer address.

The W bit gives optional auto increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base value may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base. The only use of the W bit in a post-indexed data transfer is in privileged mode code, where setting the W bit forces non-privileged mode for the transfer, allowing the operating system to generate a user address in a system where the memory management hardware makes suitable use of this hardware.

SHIFTED REGISTER OFFSET

The 8 shift control bits are described in the data processing instructions section. However, the register specified shift amounts are not available in this instruction class. See Figure 3-5.

BYTES AND WORDS

This instruction class may be used to transfer a byte (B=1) or a word (B=0) between an ARM7TDMI register and memory.

The action of LDR(B) and STR(B) instructions is influenced by the **BIGEND** control signal of ARM7TDMI core. The two possible configurations are described below.

NOTE

The KS32C6400 is configured to the big-endian format.

Little-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 7 through 0 if the supplied address is on a word boundary, on data bus inputs 15 through 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register, and the remaining bits of the register are filled with zeros. Please see Figure 2-2.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) will normally use a word aligned address. However, an address offset from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 0 to 7. This means that half-words accessed at offsets 0 and 2 from the word boundary will be correctly loaded into bits 0 through 15 of the register. Two shift operations are then required to clear or to sign extend the upper 16 bits.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

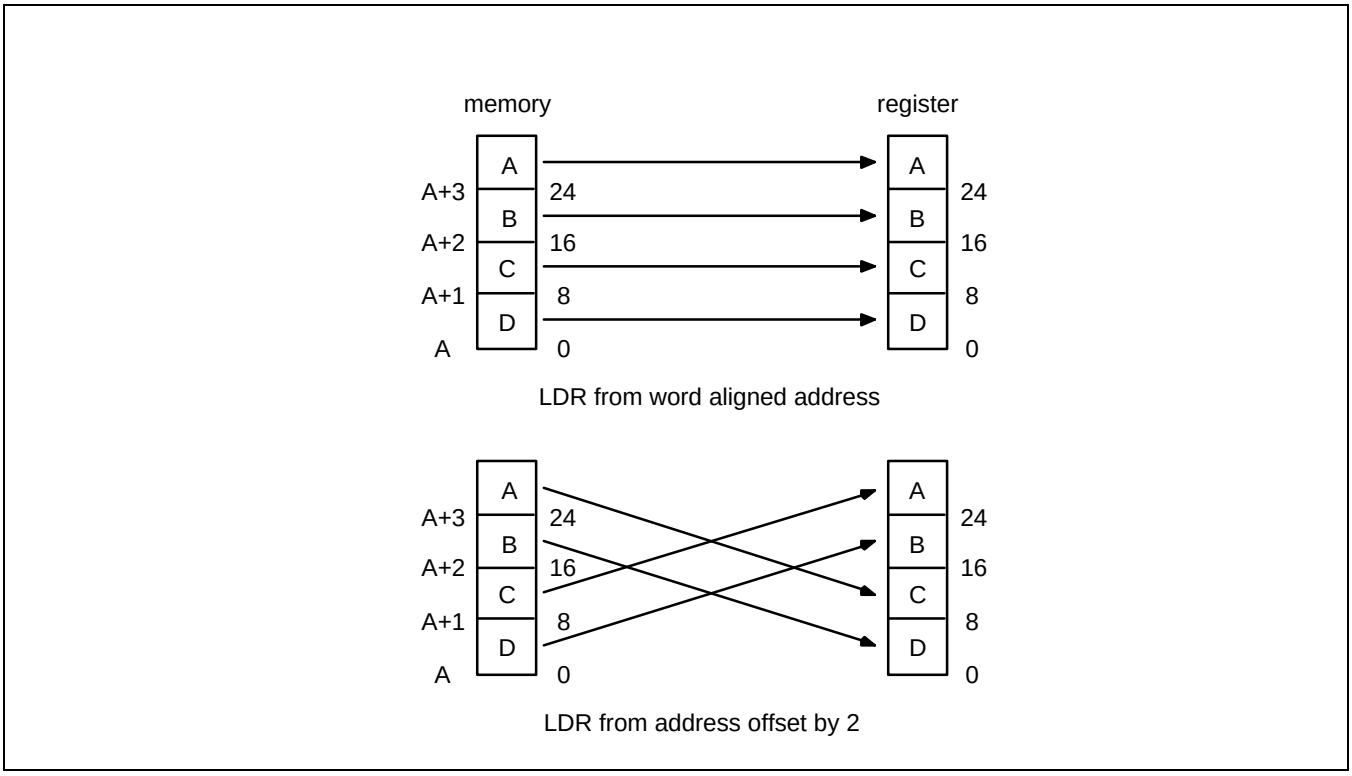


Figure 3-15. Little-Endian Offset Addressing

Big-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 31 through 24 if the supplied address is on a word boundary, on data bus inputs 23 through 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register and the remaining bits of the register are filled with zeros. Please see Figure 2-1.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) should generate a word aligned address. An address offset of 0 or 2 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 31 through 24. This means that half-words accessed at these offsets will be correctly loaded into bits 16 through 31 of the register. A shift operation is then required to move (and optionally sign extend) the data into the bottom 16 bits. An address offset of 1 or 3 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 15 through 8.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

USE OF R15

Write-back must not be specified if R15 is specified as the base register (Rn). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 must not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a register store (STR) instruction, the stored value will be address of the instruction plus 12.

RESTRICTION ON THE USE OF BASE REGISTER

When configured for late aborts, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

After an abort, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

Example:

```
LDR     R0,[R1],R1
```

Therefore a post-indexed LDR or STR where Rm is the same register as Rn should not be used.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR instructions take 1S + 1N + 1I and LDR PC take 2S + 2N +1I incremental cycles, where S,N and I are defined as squential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STR instructions take 2N incremental cycles to execute.

ASSEMBLER SYNTAX

<LDR|STR>{cond}{B}{T} Rd,<Address>

where:

LDR	Load from memory into a register
STR	Store from a register into memory
{cond}	Two-character condition mnemonic. See Table 3-2.
{B}	If B is present then byte transfer, otherwise word transfer
{T}	If T is present the W bit will be set in a post-indexed instruction, forcing non-privileged mode for the transfer cycle. T is not allowed when a pre-indexed addressing mode is specified or implied.
Rd	An expression evaluating to a valid register number.
Rn and Rm	Expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining. In this case base write-back should not be specified.

<Address>can be:

1	An expression which generates an address: The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.	
2	A pre-indexed addressing specification:	
	[Rn]	offset of zero
	[Rn,<#expression>]{!}	offset of <expression> bytes
	[Rn,{+/-}Rm{,<shift>}]{!}	offset of +/- contents of index register, shifted by <shift>
3	A post-indexed addressing specification:	
	[Rn],<#expression>	offset of <expression> bytes
	[Rn],{+/-}Rm{,<shift>}	offset of +/- contents of index register, shifted as by <shift>.
<shift>	General shift operation (see data processing instructions) but you cannot specify the shift amount by a register.	
{!}	Writes back the base register (set the W bit) if! is present.	

EXAMPLES

```
STR     R1,[R2,R4]!     ; Store R1 at R2+R4 (both of which are registers)
                        ; and write back address to R2.
STR     R1,[R2],R4      ; Store R1 at R2 and write back R2+R4 to R2.
LDR     R1,[R2,#16]     ; Load R1 from contents of R2+16, but don't write back.
LDR     R1,[R2,R3,LSL#2] ; Load R1 from contents of R2+R3*4.
LDREQB  R1,[R6,#5]      ; Conditionally load byte at R6+5 into
                        ; R1 bits 0 to 7, filling bits 8 to 31 with zeros.
STR     R1,PLACE        ; Generate PC relative offset to address PLACE.
PLACE
```

HALFWORD AND SIGNED DATA TRANSFER (LDRH/STRH/LDRSB/LDRSH)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-16.

These instructions are used to load or store half-words of data and also load sign-extended bytes or half-words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register. The result of this calculation may be written back into the base register if auto-indexing is required.

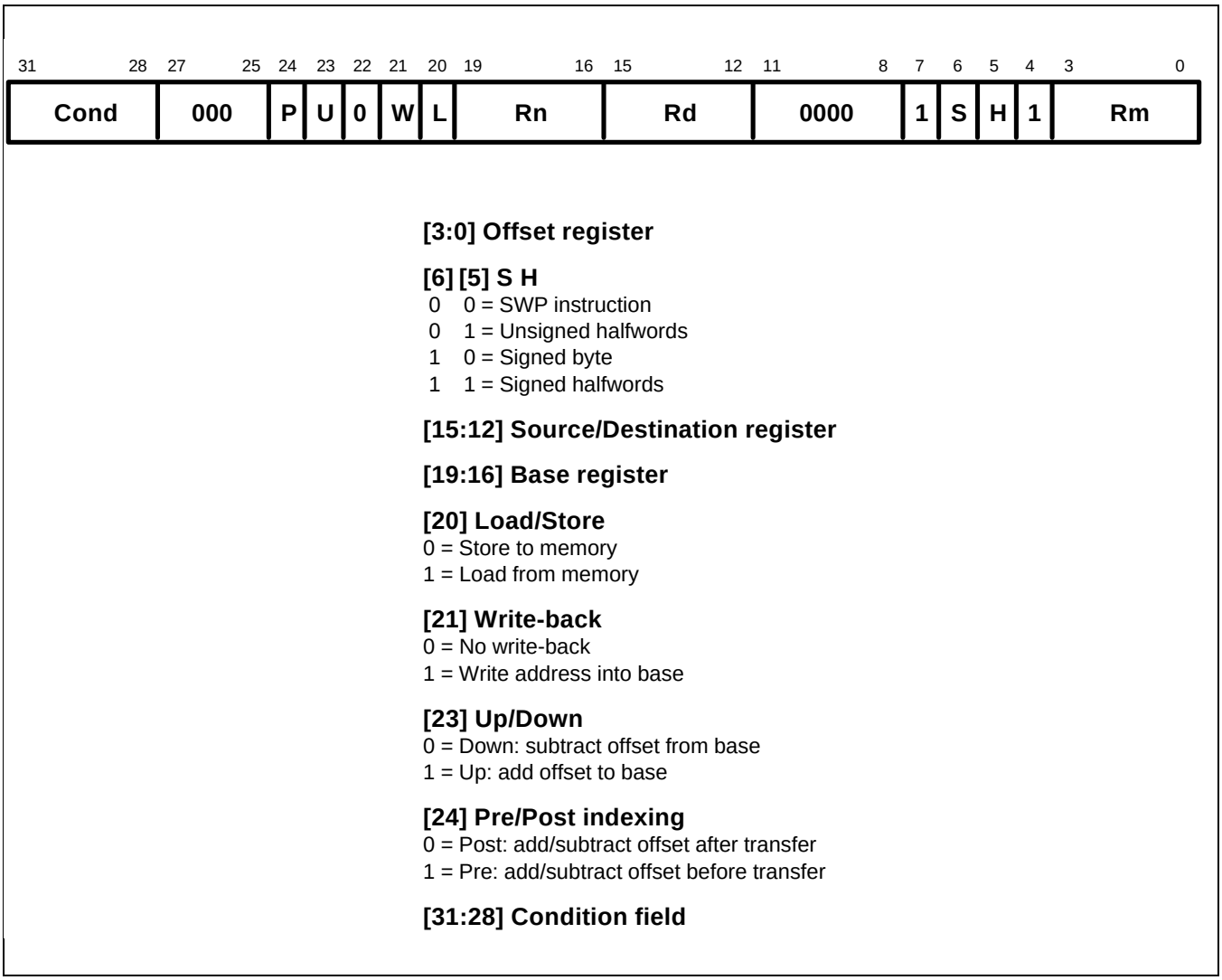
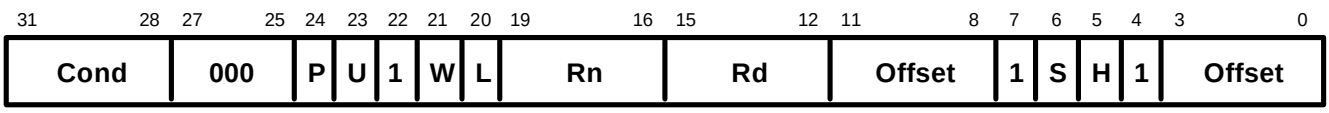


Figure 3-16. Halfword and Signed Data Transfer with Register Offset



[3:0] Immediate Offset (Low nibble)

[6] [5] S H
0 0 = SWP instruction
0 1 = Unsigned halfwords
1 0 = Signed byte
1 1 = Signed halfwords

[11:8] Immediate Offset (High nibble)

[15:12] Source/Destination register

[19:16] Base register

[20] Load/Store
0 = Store to memory
1 = Load from memory

[21] Write-back
0 = No write-back
1 = Write address into base

[23] Up/Down
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post indexing
0 = Post: add/subtract offset after transfer
1 = Pre: add/subtract offset before transfer

[31:28] Condition field

Figure 3-17. Halfword and Signed Data Transfer with Immediate Offset and Auto-Indexing

OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 8-bit unsigned binary immediate value in the instruction, or a second register. The 8-bit offset is formed by concatenating bits 11 to 8 and bits 3 to 0 of the instruction word, such that bit 11 becomes the MSB and bit 0 becomes the LSB. The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base register is used as the transfer address.

The W bit gives optional auto-increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained if necessary by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base.

The Write-back bit should not be set high (W=1) when post-indexed addressing is selected.

HALFWORD LOAD AND STORES

Setting S=0 and H=1 may be used to transfer unsigned Half-words between an ARM7TDMI register and memory.

The action of LDRH and STRH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the section below.

SIGNED BYTE AND HALFWORD LOADS

The S bit controls the loading of sign-extended data. When S=1 the H bit selects between Bytes (H=0) and Half-words (H=1). The L bit should not be set low (Store) when Signed (S=1) operations have been selected.

The LDRSB instruction loads the selected Byte into bits 7 to 0 of the destination register and bits 31 to 8 of the destination register are set to the value of bit 7, the sign bit.

The LDRSH instruction loads the selected Half-word into bits 15 to 0 of the destination register and bits 31 to 16 of the destination register are set to the value of bit 15, the sign bit.

The action of the LDRSB and LDRSH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the following section.

ENDIANNESS AND BYTE/HALFWORD SELECTION

Little-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 7 through to 0 if the supplied address is on a word boundary, on data bus inputs 15 through to 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bit of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-2.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 15 through to 0 if the supplied address is on a word boundary and on data bus inputs 31 through to 16 if it is a halfword boundary, (A[1]=1).The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH then the ARM7TDMI will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH) the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned, if bit 0 of the address is HIGH this will cause unpredictable behaviour.

Big-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 31 through to 24 if the supplied address is on a word boundary, on data bus inputs 23 through to 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bit of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-1.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 31 through to 16 if the supplied address is on a word boundary and on data bus inputs 15 through to 0 if it is a halfword boundary, (A[1]=1). The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH then the ARM7TDMI will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH) the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned, if bit 0 of the address is HIGH this will cause unpredictable behaviour.

NOTE

The KS32C6400 is configured to the big-endian format.

USE OF R15

Write-back should not be specified if R15 is specified as the base register (Rn). When using R15 as the base register you must remember it contains an address 8 bytes on from the address of the current instruction.

R15 should not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a Half-word store (STRH) instruction, the stored address will be address of the instruction plus 12.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory the required data may be absent from the main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR(H,SH,SB) instructions take 1S + 1N + 1I. LDR(H,SH,SB) PC take 2S + 2N + 1I incremental cycles. S,N and I are defined as squential (S-cycle), non-squential (N-cycle), and internal (I-cycle), respectively. STRH instructions take 2N incremental cycles to execute.

ASSEMBLER SYNTAX

<LDR|STR>{cond}<H|SH|SB> Rd,<address>

LDR	Load from memory into a register
STR	Store from a register into memory
{cond}	Two-character condition mnemonic. See Table 3-2..
H	Transfer halfword quantity
SB	Load sign extended byte (Only valid for LDR)
SH	Load sign extended halfword (Only valid for LDR)
Rd	An expression evaluating to a valid register number.

<address> can be:

1	An expression which generates an address: The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.	
2	A pre-indexed addressing specification:	
	[Rn]	offset of zero
	[Rn,<#expression>]{!}	offset of <expression> bytes
	[Rn,{+/-}Rm]{!}	offset of +/- contents of index register
3	A post-indexed addressing specification:	
	[Rn],<#expression>	offset of <expression> bytes
	[Rn],{+/-}Rm	offset of +/- contents of index register.
4	Rn and Rm are expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining. In this case base write-back should not be specified.	
{!}	Writes back the base register (set the W bit) if ! is present.	

EXAMPLES

```
        LDRH    R1,[R2,-R3]!        ;  Load R1 from the contents of the halfword address
                                    ;  contained in R2-R3 (both of which are registers)
                                    ;  and write back address to R2
        STRH    R3,[R4,#14]         ;  Store the halfword in R3 at R14+14 but don't write
back.
        LDRSB   R8,[R2],#-223       ;  Load R8 with the sign extended contents of the byte
                                    ;  address contained in R2 and write back R2-223 to R2.
        LDRNESH R11,[R0]            ;  Conditionally load R11 with the sign extended contents
of
                                    ;  the halfword address contained in R0.
        HERE                        ;  Generate PC relative offset to address FRED.
        STRH    R5, [PC,#(FRED-HERE-8)] ;  Store the halfword in R5 at address FRED
        FRED
```

BLOCK DATA TRANSFER (LDM, STM)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-18.

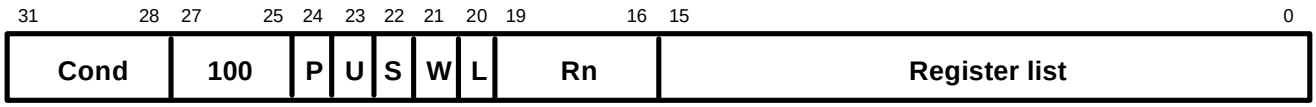
Block data transfer instructions are used to load (LDM) or store (STM) any subset of the currently visible registers. They support all possible stacking modes, maintaining full or empty stacks which can grow up or down memory, and are very efficient instructions for saving or restoring context, or for moving large blocks of data around main memory.

THE REGISTER LIST

The instruction can cause the transfer of any registers in the current bank (and non-user mode programs can also transfer to and from the user bank, see below). The register list is a 16 bit field in the instruction, with each bit corresponding to a register. A 1 in bit 0 of the register field will cause R0 to be transferred, a 0 will cause it not to be transferred; similarly bit 1 controls the transfer of R1, and so on.

Any subset of the registers, or all the registers, may be specified. The only restriction is that the register list should not be empty.

Whenever R15 is stored to memory the stored value is the address of the STM instruction plus 12.



[19:16] Base register

[20] Load/Store bit
0 = Store to memory
1 = Load from memory

[21] Write-back bit
0 = No write-back
1 = Write address into base

[[22] PSR & force user bit
0 = Do not load PSR or force user mode
1 = Load PSR or force user mode

[23] Up/Down bit
0 = Down; subtrack offset from base
1 = Up; add offset to base

[24] Pre/Post indexing bit
0 = Post; add offset after transfer
1 = Pre; add offset before transfer

[31:28] Condition field

Figure 3-18. Block Data Transfer Instructions

ADDRESSING MODES

The transfer addresses are determined by the contents of the base register (Rn), the pre/post bit (P) and the up/down bit (U). The registers are transferred in the order lowest to highest, so R15 (if in the list) will always be transferred last. The lowest register also gets transferred to/from the lowest memory address. By way of illustration, consider the transfer of R1, R5 and R7 in the case where Rn=0x1000 and write back of the modified base is required (W=1). Figure 3.19–22 show the sequence of register transfers, the addresses used, and the value of Rn after the instruction has completed.

In all cases, had write back of the modified base not been required (W=0), Rn would have retained its initial value of 0x1000 unless it was also in the transfer list of a load multiple register instruction, when it would have been overwritten with the loaded value.

ADDRESS ALIGNMENT

The address should normally be a word aligned quantity and non-word aligned addresses do not affect the instruction. However, the bottom 2 bits of the address will appear on **A[1:0]** and might be interpreted by the memory system.

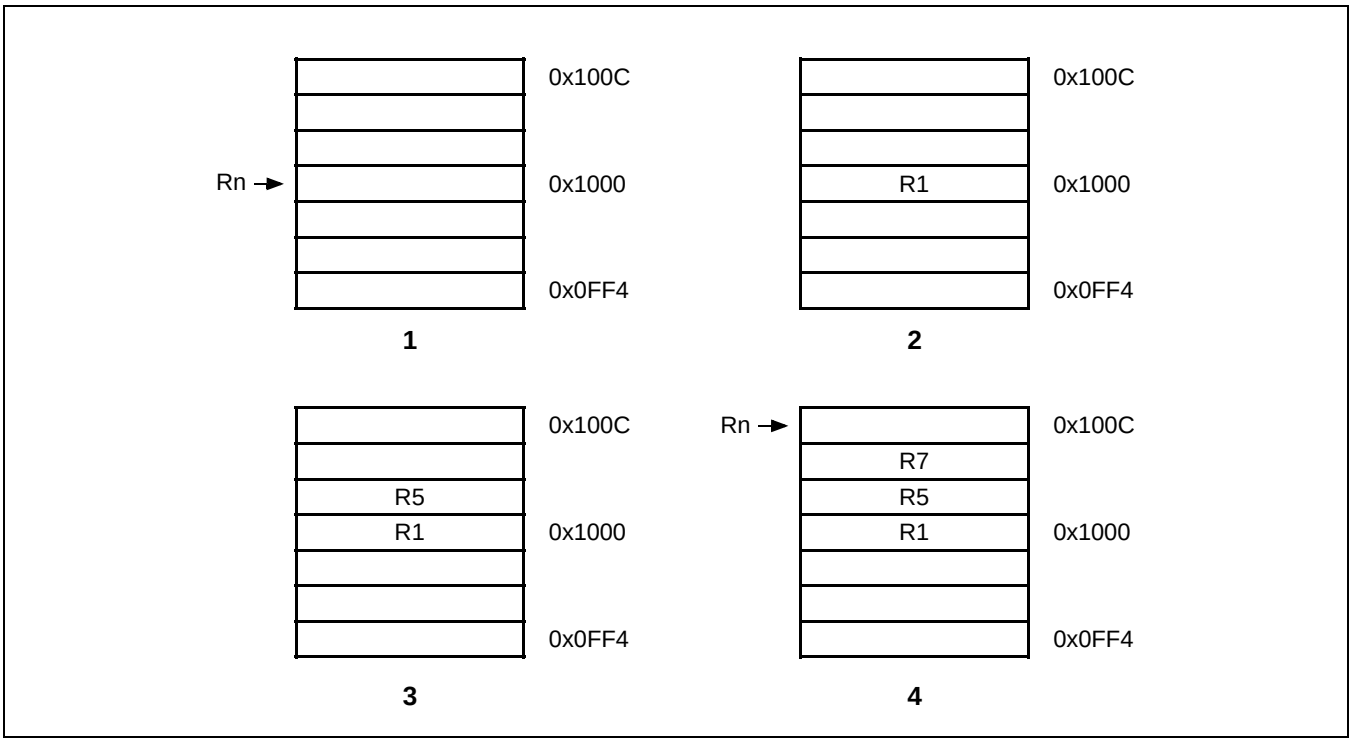


Figure 3-19. Post-Increment Addressing

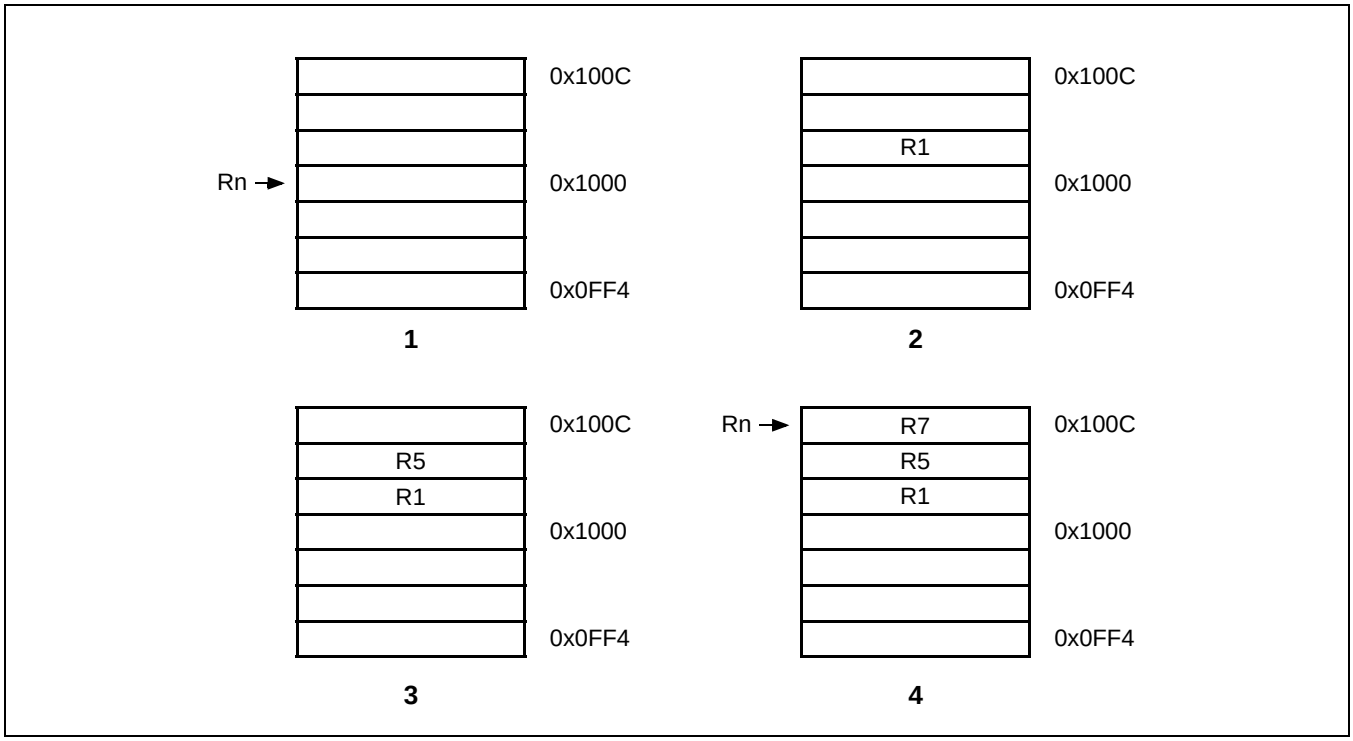


Figure 3-20. Pre-Increment Addressing

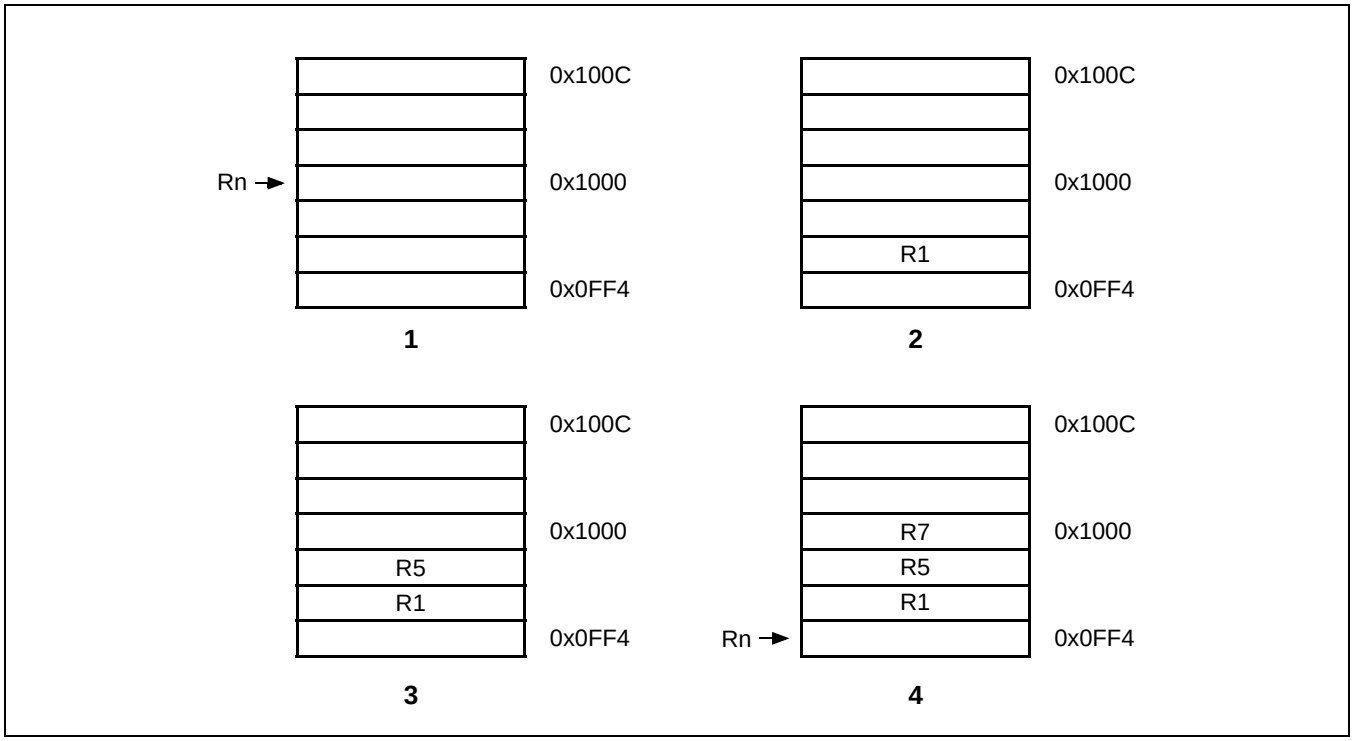


Figure 3-21. Post-Decrement Addressing

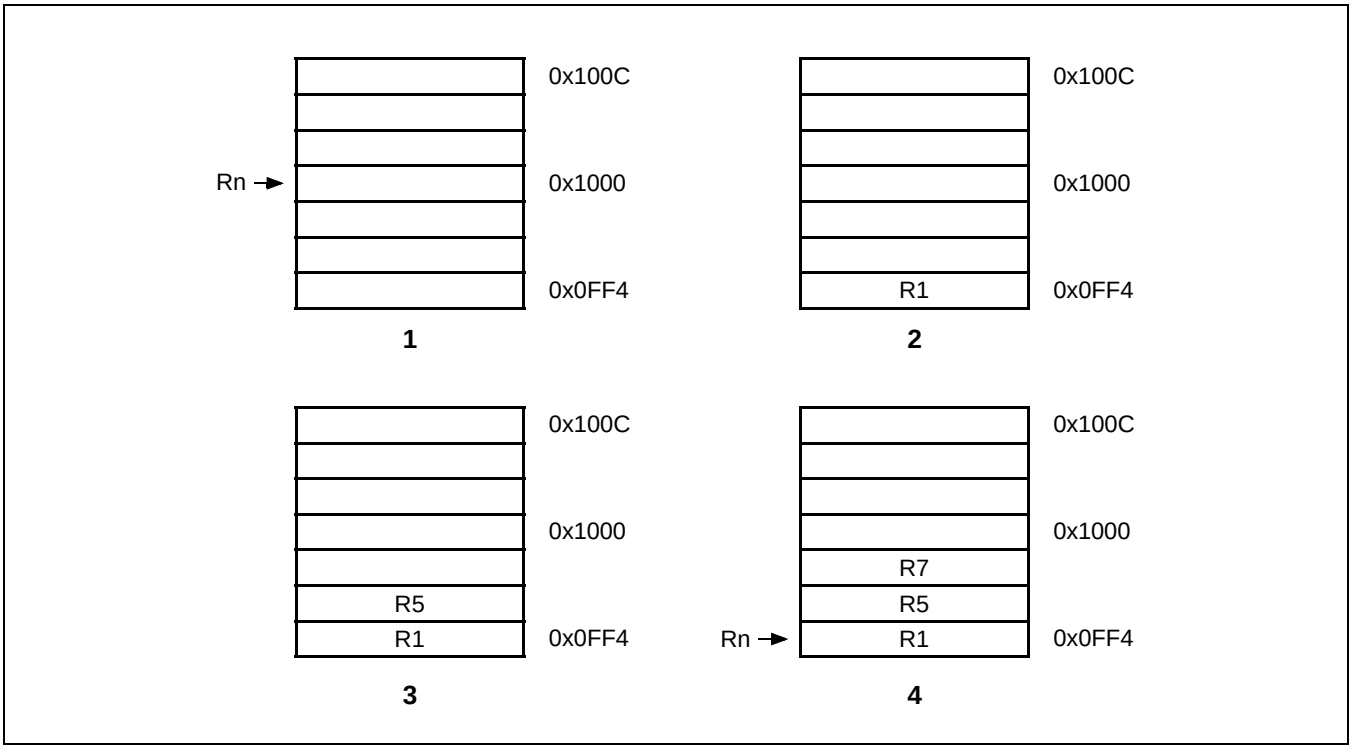


Figure 3-22. Pre-Decrement Addressing

USE OF THE S BIT

When the S bit is set in a LDM/STM instruction its meaning depends on whether or not R15 is in the transfer list and on the type of instruction. The S bit should only be set if the instruction is to execute in a privileged mode.

LDM with R15 in Transfer List and S Bit Set (Mode Changes)

If the instruction is a LDM then SPSR_<mode> is transferred to CPSR at the same time as R15 is loaded.

STM with R15 in Transfer List and S Bit Set (User Bank Transfer)

The registers transferred are taken from the User bank rather than the bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

R15 not in List and S Bit Set (User Bank Transfer)

For both LDM and STM instructions, the User bank registers are transferred rather than the register bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

When the instruction is LDM, care must be taken not to read from a banked register during the following cycle (inserting a dummy instruction such as MOV R0, R0 after the LDM will ensure safety).

USE OF R15 AS THE BASE

R15 should not be used as the base register in any LDM or STM instruction.

INCLUSION OF THE BASE IN THE REGISTER LIST

When write-back is specified, the base is written back at the end of the second cycle of the instruction. During a STM, the first register is written out at the start of the second cycle. A STM which includes storing the base, with the base as the first register to be stored, will therefore store the unchanged value, whereas with the base second or later in the transfer order, will store the modified value. A LDM will always overwrite the updated base if the base is in the list.

DATA ABORTS

Some legal addresses may be unacceptable to a memory management system, and the memory manager can indicate a problem with an address by taking the **ABORT** signal HIGH. This can happen on any transfer during a multiple register load or store, and must be recoverable if ARM7TDMI is to be used in a virtual memory system.

Aborts during STM Instructions

If the abort occurs during a store multiple instruction, ARM7TDMI takes little action until the instruction completes, whereupon it enters the data abort trap. The memory manager is responsible for preventing erroneous writes to the memory. The only change to the internal state of the processor will be the modification of the base register if write-back was specified, and this must be reversed by software (and the cause of the abort resolved) before the instruction may be retried.

Aborts during LDM Instructions

When ARM7TDMI detects a data abort during a load multiple instruction, it modifies the operation of the instruction to ensure that recovery is possible.

- Overwriting of registers stops when the abort happens. The aborting load will not take place but earlier ones may have overwritten registers. The PC is always the last register to be written and so will always be preserved.
- The base register is restored, to its modified value if write-back was requested. This ensures recoverability in the case where the base register is also in the transfer list, and may have been overwritten before the abort occurred.

The data abort trap is taken when the load multiple has completed, and the system software must undo any base modification (and resolve the cause of the abort) before restarting the instruction.

INSTRUCTION CYCLE TIMES

Normal LDM instructions take nS + 1N + 1I and LDM PC takes (n+1)S + 2N + 1I incremental cycles, where S,N and I are defined as squential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STM instructions take (n-1)S + 2N incremental cycles to execute, where *n* is the number of words transferred.

ASSEMBLER SYNTAX

<LDM|STM>{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!},<Rlist>{^}

where:

{cond}	Two character condition mnemonic. See Table 3-2.
Rn	An expression evaluating to a valid register number
<Rlist>	A list of registers and register ranges enclosed in {} (e.g. {R0,R2-R7,R10}).
{!}	If present requests write-back (W=1), otherwise W=0.
{^}	If present set S bit to load the CPSR along with the PC, or force transfer of user bank when in privileged mode.

Addressing Mode Names

There are different assembler mnemonics for each of the addressing modes, depending on whether the instruction is being used to support stacks or for other purposes. The equivalence between the names and the values of the bits in the instruction are shown in the following table 3-6.

Table 3-6. Addressing Mode Names

Name	Stack	Other	L bit	P bit	U bit
Pre-Increment Load	LDMED	LDMIB	1	1	1
Post-Increment Load	LDMFD	LDMIA	1	0	1
Pre-Dcrement Load	LDMEA	LDMDB	1	1	0
Post-Decrement Load	LDMFA	LDMDA	1	0	0
Pre-Increment Store	STMFA	STMIB	0	1	1
Post-Increment Store	STMEA	STMIA	0	0	1
Pre-Decrement Store	STMFD	STMDB	0	1	0
Post-Decrement Store	STMED	STMDA	0	0	0

FD, ED, FA, EA define pre/post indexing and the up/down bit by reference to the form of stack required. The F and E refer to a “full” or “empty” stack, i.e. whether a pre-index has to be done (full) before storing to the stack. The A and D refer to whether the stack is ascending or descending. If ascending, a STM will go up and LDM down, if descending, vice-versa.

IA, IB, DA, DB allow control when LDM/STM are not being used for stacks and simply mean Increment After, Increment Before, Decrement After, Decrement Before.

EXAMPLES

```
LDMFD   SP!,{R0,R1,R2}      ; Unstack 3 registers.
STMIA   R0,{R0-R15}         ; Save all registers.
LDMFD   SP!,{R15}           ; R15 <- (SP), CPSR unchanged.
LDMFD   SP!,{R15}^          ; R15 <- (SP), CPSR <- SPSR_mode
                            ; (allowed only in privileged modes).
STMFD   R13,{R0-R14}^       ; Save user mode regs on stack
                            ; (allowed only in privileged modes).
```

These instructions may be used to save state on subroutine entry, and restore it efficiently on return to the calling routine:

```
STMED   SP!,{R0-R3,R14}     ; Save R0 to R3 to use as workspace
                            ; and R14 for returning.
BL      somewhere           ; This nested call will overwrite R14
LDMED   SP!,{R0-R3,R15}     ; Restore workspace and return.
```

SINGLE DATA SWAP (SWP)

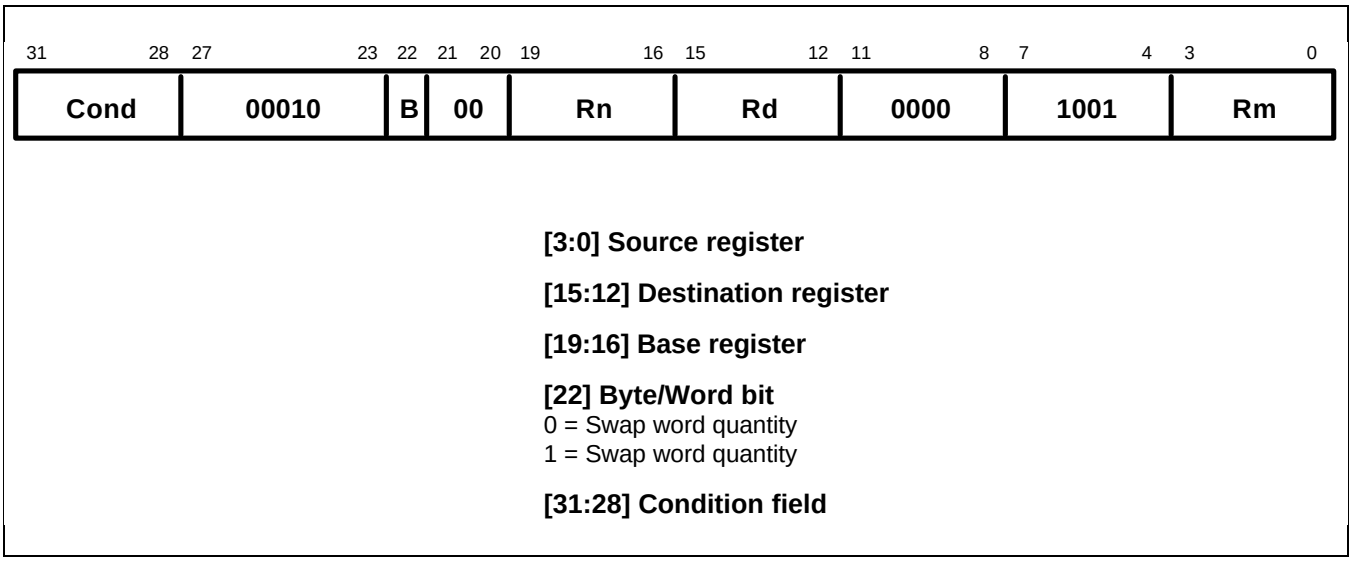


Figure 3-23. Swap Instruction

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-23.

The data swap instruction is used to swap a byte or word quantity between a register and external memory. This instruction is implemented as a memory read followed by a memory write which are “locked” together (the processor cannot be interrupted until both operations have completed, and the memory manager is warned to treat them as inseparable). This class of instruction is particularly useful for implementing software semaphores.

The swap address is determined by the contents of the base register (Rn). The processor first reads the contents of the swap address. Then it writes the contents of the source register (Rm) to the swap address, and stores the old memory contents in the destination register (Rd). The same register may be specified as both the source and destination.

The **LOCK** output goes HIGH for the duration of the read and write operations to signal to the external memory manager that they are locked together, and should be allowed to complete without interruption. This is important in multi-processor systems where the swap instruction is the only indivisible instruction which may be used to implement semaphores; control of the memory must not be removed from a processor while it is performing a locked operation.

BYTES AND WORDS

This instruction class may be used to swap a byte (B=1) or a word (B=0) between an ARM7TDMI register and memory. The SWP instruction is implemented as a LDR followed by a STR and the action of these is as described in the section on single data transfers. In particular, the description of Big and Little Endian configuration applies to the SWP instruction.

USE OF R15

Do not use R15 as an operand (Rd, Rn or Rs) in a SWP instruction.

DATA ABORTS

If the address used for the swap is unacceptable to a memory management system, the memory manager can flag the problem by driving ABORT HIGH. This can happen on either the read or the write cycle (or both), and in either case, the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Swap instructions take 1S + 2N +1I incremental cycles to execute, where S,N and I are defined as squential (S-cycle), non-squential, and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

<SWP>{cond}{B} Rd,Rm,[Rn]

{cond}	Two-character condition mnemonic. See Table 3-2.
{B}	If B is present then byte transfer, otherwise word transfer
Rd,Rm,Rn	Expressions evaluating to valid register numbers

EXAMPLES

```
SWP     R0,R1,[R2]      ; Load R0 with the word addressed by R2, and
                        ; store R1 at R2.
SWPB    R2,R3,[R4]      ; Load R2 with the byte addressed by R4, and
                        ; store bits 0 to 7 of R3 at R4.
SWPEQ   R0,R0,[R1]      ; Conditionally swap the contents of the
                        ; word addressed by R1 with R0.
```

SOFTWARE INTERRUPT (SWI)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-24, below.

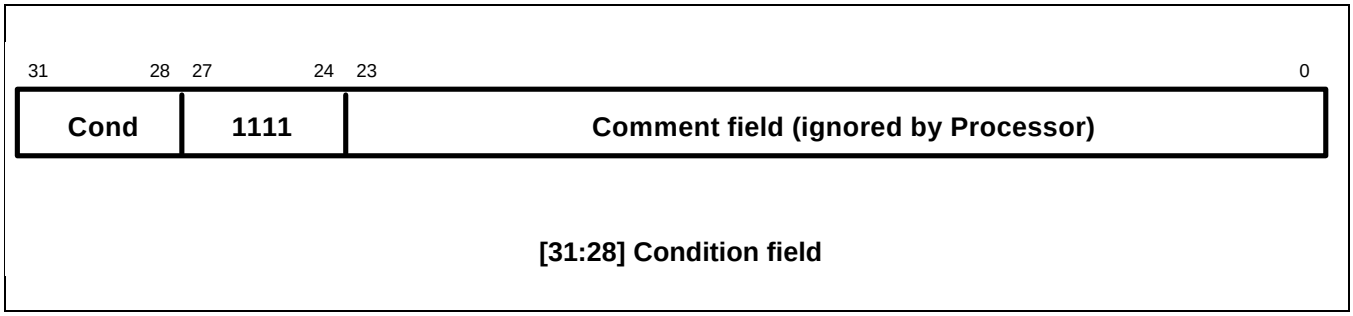


Figure 3-24. Software Interrupt Instruction

The software interrupt instruction is used to enter Supervisor mode in a controlled manner. The instruction causes the software interrupt trap to be taken, which effects the mode change. The PC is then forced to a fixed value (0x08) and the CPSR is saved in SPSR_svc. If the SWI vector address is suitably protected (by external memory management hardware) from modification by the user, a fully protected operating system may be constructed.

RETURN FROM THE SUPERVISOR

The PC is saved in R14_svc upon entering the software interrupt trap, with the PC adjusted to point to the word after the SWI instruction. MOVS PC,R14_svc will return to the calling program and restore the CPSR.

Note that the link mechanism is not re-entrant, so if the supervisor code wishes to use software interrupts within itself it must first save a copy of the return address and SPSR.

COMMENT FIELD

The bottom 24 bits of the instruction are ignored by the processor, and may be used to communicate information to the supervisor code. For instance, the supervisor may look at this field and use it to index into an array of entry points for routines which perform the various supervisor functions.

INSTRUCTION CYCLE TIMES

Software interrupt instructions take 2S + 1N incremental cycles to execute, where S and N are defined as squential (S-cycle) and non-squential (N-cycle).

ASSEMBLER SYNTAX

SWI{cond} <expression>

{cond}	Two character condition mnemonic, Table 3-2.
<expression>	Evaluated and placed in the comment field (which is ignored by ARM7TDMI).

EXAMPLES

```
        SWI     ReadC           ;  Get next character from read stream.
        SWI     WriteI+"k"      ;  Output a "k" to the write stream.
        SWINE   0               ;  Conditionally call supervisor with 0 in comment field.
```

Supervisor code

The previous examples assume that suitable supervisor code exists, for instance:

```
        0x08 B Supervisor               ;  SWI entry point
        EntryTable                      ;  Addresses of supervisor routines
        DCD ZeroRtn
        DCD ReadCRtn
        DCD WriteIRtn
        . . .
Zero    EQU  0
ReadC   EQU  256
WriteI  EQU  512

        Supervisor                      ;  SWI has routine required in bits 8-23 and data (if any)
in
                                        ;  bits 0–7. Assumes R13_svc points to a suitable stack
        STMFD   R13,{R0-R2,R14}         ;  Save work registers and return address.
        LDR     R0,[R14,#-4]            ;  Get SWI instruction.
        BIC     R0,R0,#0xFF000000       ;  Clear top 8 bits.
        MOV     R1,R0,LSR#8             ;  Get routine offset.
        ADR     R2,EntryTable           ;  Get start address of entry table.
        LDR     R15,[R2,R1,LSL#2]       ;  Branch to appropriate routine.
        WriteIRtn                       ;  Enter with character in R0 bits 0–7.
        . . . . . .
        LDMFD   R13,{R0-R2,R15}^        ;  Restore workspace and return,
                                        ;  restoring processor mode and flags.
```

COPROCESSOR DATA OPERATIONS (CDP)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-25.

This class of instruction is used to tell a coprocessor to perform some internal operation. No result is communicated back to ARM7TDMI, and it will not wait for the operation to complete. The coprocessor could contain a queue of such instructions awaiting execution, and their execution can overlap other activity, allowing the coprocessor and ARM7TDMI to perform independent tasks in parallel.

COPROCESSOR INSTRUCTIONS

The KS32C6400, unlike some other ARM-based processors, does not have an external coprocessor interface. It does not have a on-chip coprocessor also.

So then all coprocessor instructions will cause the undefinded instruction trap to be taken on the KS32C6400. These coprocessor instructions can be emulated by the undefined trap handler. Even though external coprocessor can not be connected to the KS32C6400, the coprocessor instructions are still described here in full for completeness. (Remember that any external coprocessor described in this section is a software emulation.)

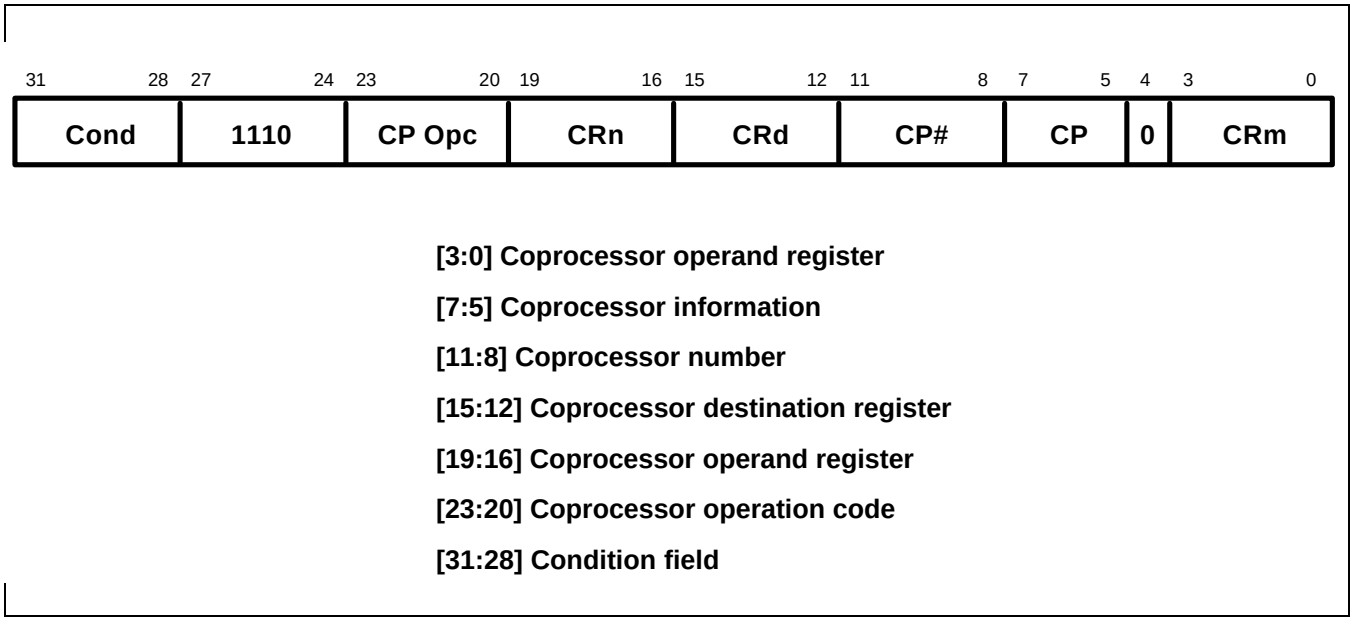


Figure 3-25. Coprocessor Data Operation Instruction

THE COPROCESSOR FIELDS

Only bit 4 and bits 24 to 31 are significant to ARM7TDMI. The remaining bits are used by coprocessors. The above field names are used by convention, and particular coprocessors may redefine the use of all fields except CP# as appropriate. The CP# field is used to contain an identifying number (in the range 0 to 15) for each coprocessor, and a coprocessor will ignore any instruction which does not contain its number in the CP# field.

The conventional interpretation of the instruction is that the coprocessor should perform an operation specified in the CP Opc field (and possibly in the CP field) on the contents of CRn and CRm, and place the result in CRd.

INSTRUCTION CYCLE TIMES

Coprocessor data operations take 1S + bI incremental cycles to execute, where *b* is the number of cycles spent in the coprocessor busy-wait loop.

S and I are defined as squential (S-cycle) and internal (I-cycle).

ASSEMBLER SYNTAX

CDP{cond} p#,<expression1>,cd,cn,cm{,<expression2>}

{cond}	Two character condition mnemonic. See Table 3-2.
p#	The unique number of the required coprocessor
<expression1>	Evaluated to a constant and placed in the CP Opc field
cd, cn and cm	Evaluate to the valid coprocessor register numbers CRd, CRn and CRm respectively
<expression2>	Where present is evaluated to a constant and placed in the CP field

EXAMPLES

```
        CDP     p1,10,c1,c2,c3      ; Request coproc 1 to do operation 10
                                    ; on CR2 and CR3, and put the result in CR1.
        CDPEQ   p2,5,c1,c2,c3,2     ; If Z flag is set request coproc 2 to do operation 5 (type 2)
                                    ; on CR2 and CR3, and put the result in CR1.
```

COPROCESSOR DATA TRANSFERS (LDC, STC)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction encoding is shown in Figure 3-26.

This class of instruction is used to load (LDC) or store (STC) a subset of a coprocessors's registers directly to memory. ARM7TDMI is responsible for supplying the memory address, and the coprocessor supplies or accepts the data and controls the number of words transferred.

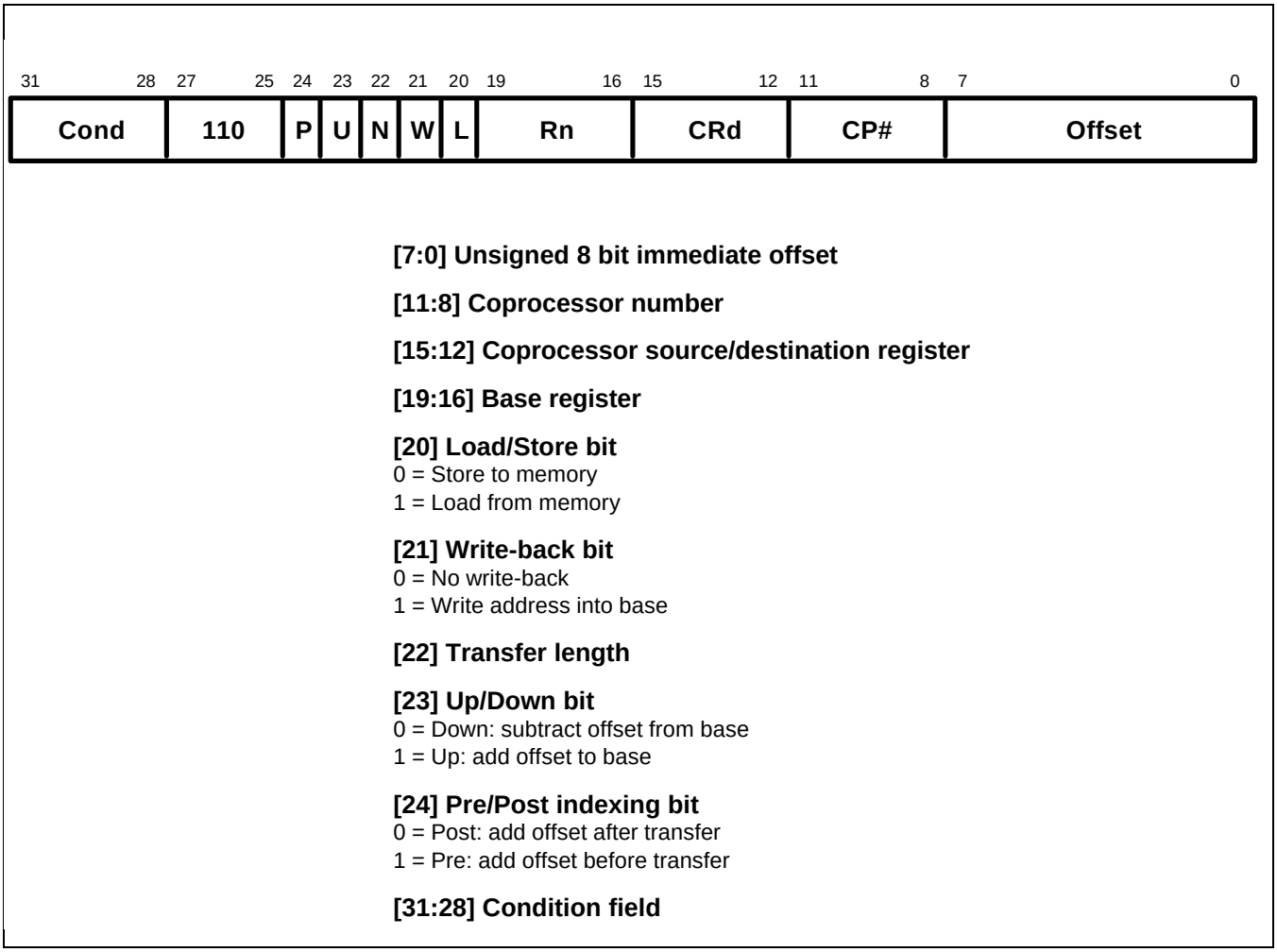


Figure 3-26. Coprocessor Data Transfer Instructions

THE COPROCESSOR FIELDS

The CP# field is used to identify the coprocessor which is required to supply or accept the data, and a coprocessor will only respond if its number matches the contents of this field.

The CRd field and the N bit contain information for the coprocessor which may be interpreted in different ways by different coprocessors, but by convention CRd is the register to be transferred (or the first register where more than one is to be transferred), and the N bit is used to choose one of two transfer length options. For instance N=0 could select the transfer of a single register, and N=1 could select the transfer of all the registers for context switching.

ADDRESSING MODES

ARM7TDMI is responsible for providing the address used by the memory system for the transfer, and the addressing modes available are a subset of those used in single data transfer instructions. Note, however, that the immediate offsets are 8 bits wide and specify word offsets for coprocessor data transfers, whereas they are 12 bits wide and specify byte offsets for single data transfers.

The 8 bit unsigned immediate offset is shifted left 2 bits and either added to (U=1) or subtracted from (U=0) the base register (Rn); this calculation may be performed either before (P=1) or after (P=0) the base is used as the transfer address. The modified base value may be overwritten back into the base register (if W=1), or the old value of the base may be preserved (W=0). Note that post-indexed addressing modes require explicit setting of the W bit, unlike LDR and STR which always write-back when post-indexed.

The value of the base register, modified by the offset in a pre-indexed instruction, is used as the address for the transfer of the first word. The second word (if more than one is transferred) will go to or come from an address one word (4 bytes) higher than the first transfer, and the address will be incremented by one word for each subsequent transfer.

ADDRESS ALIGNMENT

The base address should normally be a word aligned quantity. The bottom 2 bits of the address will appear on **A[1:0]** and might be interpreted by the memory system.

USE OF R15

If Rn is R15, the value used will be the address of the instruction plus 8 bytes. Base write-back to R15 must not be specified.

DATA ABORTS

If the address is legal but the memory manager generates an abort, the data trap will be taken. The write-back of the modified base will take place, but all other processor state will be preserved. The coprocessor is partly responsible for ensuring that the data transfer can be restarted after the cause of the abort has been resolved, and must ensure that any subsequent actions it undertakes can be repeated when the instruction is retried.

INSTRUCTION CYCLE TIMES

Coprocessor data transfer instructions take (n-1)S + 2N + bI incremental cycles to execute, where:

n	The number of words transferred.
b	The number of cycles spent in the coprocessor busy-wait loop.

S, N and I are defined as squential (S-cycle), non-squential (N-cycle), and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

<LDC|STC>{cond}{L} p#,cd,<Address>

LDC	Load from memory to coprocessor
STC	Store from coprocessor to memory
{L}	When present perform long transfer (N=1), otherwise perform short transfer (N=0)
{cond}	Two character condition mnemonic. See Table 3-2..
p#	The unique number of the required coprocessor
cd	An expression evaluating to a valid coprocessor register number that is placed in the CRd field

<Address> can be:

1	An expression which generates an address: The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated	
2	A pre-indexed addressing specification:	
	[Rn]	offset of zero
	[Rn,<#expression>]{!}	offset of <expression> bytes
3	A post-indexed addressing specification:	
	Rn],<#expression	offset of <expression> bytes
	{!}	write back the base register (set the W bit) if! is present
	Rn	is an expression evaluating to a valid ARM7TDMI register number.

NOTE

If Rn is R15, the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining.

EXAMPLES

```
        LDC     p1,c2,table     ;  Load c2 of coproc 1 from address
                                ;  table, using a PC relative address.
        STCEQL  p2,c3,[R5,#24]! ;  Conditionally store c3 of coproc 2
                                ;  into an address 24 bytes up from R5,
                                ;  write this address back to R5, and use
                                ;  long transfer option (probably to store multiple words).
```

NOTE

Although the address offset is expressed in bytes, the instruction offset field is in words. The assembler will adjust the offset appropriately.

COPROCESSOR REGISTER TRANSFERS (MRC, MCR)

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2.. The instruction encoding is shown in Figure 3-27.

This class of instruction is used to communicate information directly between ARM7TDMI and a coprocessor. An example of a coprocessor to ARM7TDMI register transfer (MRC) instruction would be a FIX of a floating point value held in a coprocessor, where the floating point number is converted into a 32 bit integer within the coprocessor, and the result is then transferred to ARM7TDMI register. A FLOAT of a 32 bit value in ARM7TDMI register into a floating point value within the coprocessor illustrates the use of ARM7TDMI register to coprocessor transfer (MCR).

An important use of this instruction is to communicate control information directly from the coprocessor into the ARM7TDMI CPSR flags. As an example, the result of a comparison of two floating point values within a coprocessor can be moved to the CPSR to control the subsequent flow of execution.

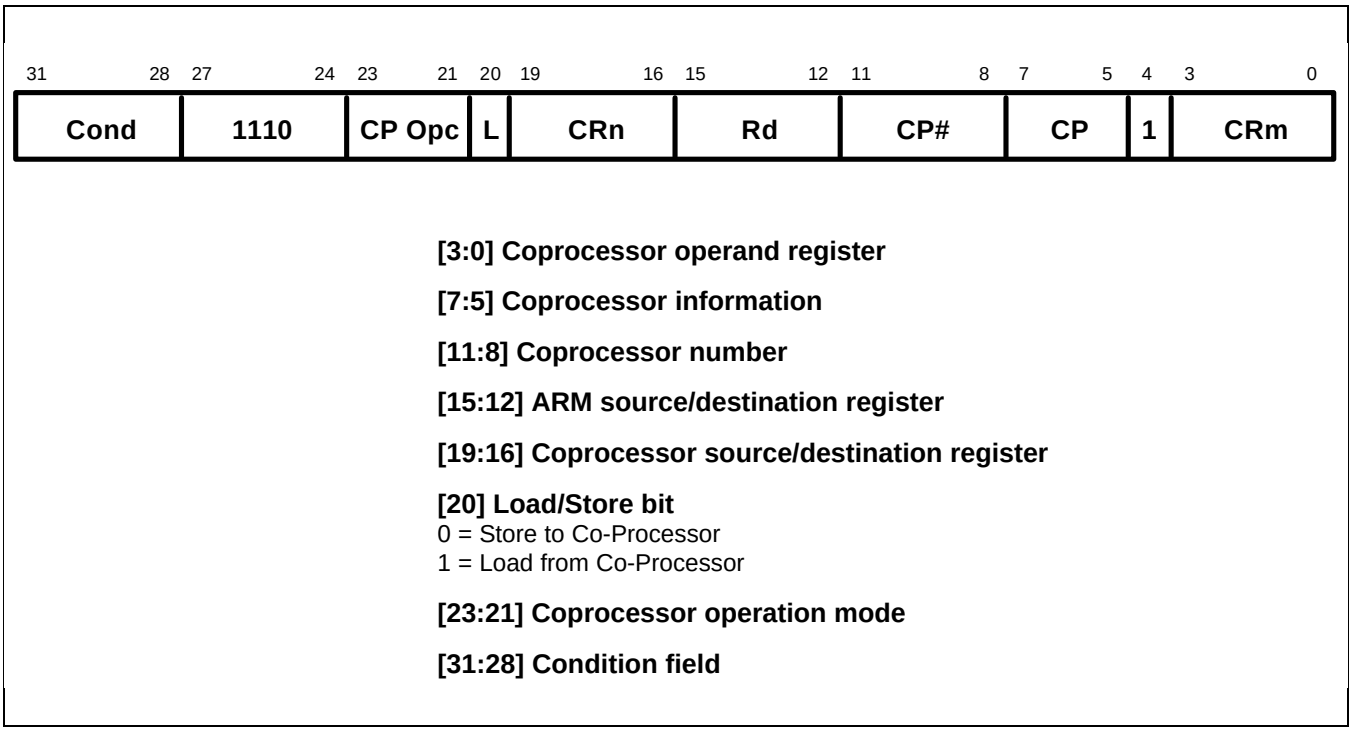


Figure 3-27. Coprocesspr Register Transfer Instructions

THE COPROCESSOR FIELDS

The CP# field is used, as for all coprocessor instructions, to specify which coprocessor is being called upon.

The CP Opc, CRn, CP and CRm fields are used only by the coprocessor, and the interpretation presented here is derived from convention only. Other interpretations are allowed where the coprocessor functionality is incompatible with this one. The conventional interpretation is that the CP Opc and CP fields specify the operation the coprocessor is required to perform, CRn is the coprocessor register which is the source or destination of the transferred information, and CRm is a second coprocessor register which may be involved in some way which depends on the particular operation specified.

TRANSFERS TO R15

When a coprocessor register transfer to ARM7TDMI has R15 as the destination, bits 31, 30, 29 and 28 of the transferred word are copied into the N, Z, C and V flags respectively. The other bits of the transferred word are ignored, and the PC and other CPSR bits are unaffected by the transfer.

TRANSFERS FROM R15

A coprocessor register transfer from ARM7TDMI with R15 as the source register will store the PC+12.

INSTRUCTION CYCLE TIMES

MRC instructions take 1S + (b+1)I +1C incremental cycles to execute, where S, I and C are defined as squential (S-cycle), internal (I-cycle), and coprocessor register transfer (C-cycle), respectively. MCR instructions take 1S + bI +1C incremental cycles to execute, where *b* is the number of cycles spent in the coprocessor busy-wait loop.

ASSEMBLER SYNTAX

<MCR|MRC>{cond} p#,<expression1>,Rd,cn,cm{,<expression2>}

MRC	Move from coprocessor to ARM7TDMI register (L=1)
MCR	Move from ARM7TDMI register to coprocessor (L=0)
{cond}	Two character condition mnemonic. See Table 3-2
p#	The unique number of the required coprocessor
<expression1>	Evaluated to a constant and placed in the CP Opc field
Rd	An expression evaluating to a valid ARM7TDMI register number
cn and cm	Expressions evaluating to the valid coprocessor register numbers CRn and CRm respectively
<expression2>	Where present is evaluated to a constant and placed in the CP field

EXAMPLES

```
        MRC     p2,5,R3,c5,c6       ; Request coproc 2 to perform operation 5
                                    ; on c5 and c6, and transfer the (single
                                    ; 32-bit word) result back to R3.
        MCR     p6,0,R4,c5,c6       ; Request coproc 6 to perform operation 0
                                    ; on R4 and place the result in c6.
        MRCEQ   p3,9,R3,c5,c6,2     ; Conditionally request coproc 3 to
                                    ; perform operation 9 (type 2) on c5 and
                                    ; c6, and transfer the result back to R3.
```

UNDEFINED INSTRUCTION

The instruction is only executed if the condition is true. The various conditions are defined in Table 3-2. The instruction format is shown in Figure 3-28.

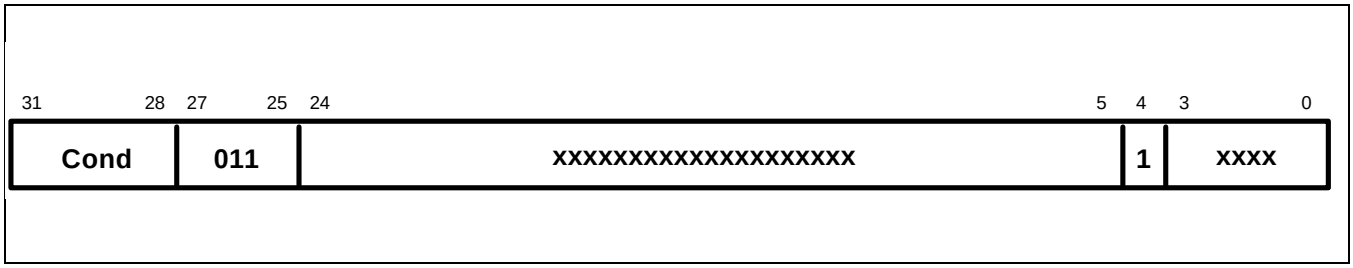


Figure 3-28. Undefined Instruction

If the condition is true, the undefined instruction trap will be taken.

Note that the undefined instruction mechanism involves offering this instruction to any coprocessors which may be present, and all coprocessors must refuse to accept it by driving **CPA** and **CPB** HIGH.

INSTRUCTION CYCLE TIMES

This instruction takes 2S + 1I + 1N cycles, where S, N and I are defined as squential (S-cycle), non-sequential (N-cycle), and internal (I-cycle).

ASSEMBLER SYNTAX

The assembler has no mnemonics for generating this instruction. If it is adopted in the future for some specified use, suitable mnemonics will be added to the assembler. Until such time, this instruction must not be used.

INSTRUCTION SET EXAMPLES

The following examples show ways in which the basic ARM7TDMI instructions can combine to give efficient code. None of these methods saves a great deal of execution time (although they may save some), mostly they just save code.

USING THE CONDITIONAL INSTRUCTIONS

Using Conditionals for Logical OR

```
        CMP     Rn,#p           ;  If Rn=p OR Rm=q THEN GOTO Label.
        BEQ     Label
        CMP     Rm,#q
        BEQ     Label
```

This can be replaced by

```
        CMP     Rn,#p
        CMPNE   Rm,#q           ;  If condition not satisfied try other test.
        BEQ     Label
```

Absolute Value

```
        TEQ     Rn,#0           ;  Test sign
        RSBMI   Rn,Rn,#0        ;  and 2's complement if necessary.
```

Multiplication by 4, 5 or 6 (Run Time)

```
        MOV     Rc,Ra,LSL#2     ;  Multiply by 4,
        CMP     Rb,#5           ;  Test value,
        ADDCS   Rc,Rc,Ra        ;  Complete multiply by 5,
        ADDHI   Rc,Rc,Ra        ;  Complete multiply by 6.
```

Combining Discrete and Range Tests

```
TEQ     Rc,#127                 ;  Discrete test,
CMPNE   Rc,#" "-1               ;  Range test
MOVLS   Rc,#"."                 ;  IF Rc<=" " OR Rc=ASCII(127)
                                ;  THEN Rc:="."
```

Division and Remainder

A number of divide routines for specific applications are provided in source form as part of the ANSI C library provided with the ARM Cross Development Toolkit, available from your supplier. A short general purpose divide routine follows.

```
                                        ;  Enter with numbers in Ra and Rb.
        MOV     Rcnt,#1                 ;  Bit to control the division.
Div1    CMP     Rb,#0x80000000          ;  Move Rb until greater than Ra.
        CMPCC   Rb,Ra
        MOVCC   Rb,Rb,ASL#1
        MOVCC   Rcnt,Rcnt,ASL#1
        BCC     Div1
        MOV     Rc,#0
Div2    CMP     Ra,Rb                   ;  Test for possible subtraction.
        SUBCS   Ra,Ra,Rb                ;  Subtract if ok,
        ADDCS   Rc,Rc,Rcnt              ;  Put relevant bit into result
        MOVS    Rcnt,Rcnt,LSR#1         ;  Shift control bit
        MOVNE   Rb,Rb,LSR#1             ;  Halve unless finished.
        BNE     Div2                    ;  Divide result in Rc, remainder in Ra.
```

Overflow Detection in the ARM7TDMI

1. Overflow in unsigned multiply with a 32-bit result

```
        UMULL   Rd,Rt,Rm,Rn             ;  3 to 6 cycles
        TEQ     Rt,#0                   ;  +1 cycle and a register
        BNE     overflow
```

2. Overflow in signed multiply with a 32-bit result

```
        SMULL   Rd,Rt,Rm,Rn             ;  3 to 6 cycles
        TEQ     Rt,Rd ASR#31            ;  +1 cycle and a register
        BNE     overflow
```

3. Overflow in unsigned multiply accumulate with a 32 bit result

```
        UMLAL   Rd,Rt,Rm,Rn             ;  4 to 7 cycles
        TEQ     Rt,#0                   ;  +1 cycle and a register
        BNE     overflow
```

4. Overflow in signed multiply accumulate with a 32 bit result

```
        SMLAL   Rd,Rt,Rm,Rn             ;  4 to 7 cycles
        TEQ     Rt,Rd, ASR#31           ;  +1 cycle and a register
        BNE     overflow
```

5. Overflow in unsigned multiply accumulate with a 64 bit result

```
UMULL   Rl,Rh,Rm,Rn     ;  3 to 6 cycles
ADDS    Rl,Rl,Ra1       ;  Lower accumulate
ADC     Rh,Rh,Ra2       ;  Upper accumulate
BCS     overflow        ;  1 cycle and 2 registers
```

6. Overflow in signed multiply accumulate with a 64 bit result

```
SMULL   Rl,Rh,Rm,Rn     ;  3 to 6 cycles
ADDS    Rl,Rl,Ra1       ;  Lower accumulate
ADC     Rh,Rh,Ra2       ;  Upper accumulate
BVS     overflow        ;  1 cycle and 2 registers
```

NOTE

Overflow checking is not applicable to unsigned and signed multiplies with a 64-bit result, since overflow does not occur in such calculations.

PSEUDO-RANDOM BINARY SEQUENCE GENERATOR

It is often necessary to generate (pseudo-) random numbers and the most efficient algorithms are based on shift generators with exclusive-OR feedback rather like a cyclic redundancy check generator. Unfortunately the sequence of a 32 bit generator needs more than one feedback tap to be maximal length (i.e. 2^32-1 cycles before repetition), so this example uses a 33 bit register with taps at bits 33 and 20. The basic algorithm is newbit:=bit 33 eor bit 20, shift left the 33 bit number and put in newbit at the bottom; this operation is performed for all the newbits needed (i.e. 32 bits). The entire operation can be done in 5 S cycles:

```
                        ;  Enter with seed in Ra (32 bits),
                        ;  Rb (1 bit in Rb lsb), uses Rc.
TST     Rb,Rb,LSR#1     ;  Top bit into carry
MOVS    Rc,Ra,RRX       ;  33 bit rotate right
ADC     Rb,Rb,Rb        ;  Carry into lsb of Rb
EOR     Rc,Rc,Ra,LSL#12 ;  (involved!)
EOR     Ra,Rc,Rc,LSR#20 ;  (similarly involved!) new seed in Ra, Rb as before
```

MULTIPLICATION BY CONSTANT USING THE BARREL SHIFTER

Multiplication by 2^n (1,2,4,8,16,32..)

```
MOV     Ra, Rb, LSL #n
```

Multiplication by 2^n+1 (3,5,9,17..)

```
ADD     Ra,Ra,Ra,LSL #n
```

Multiplication by 2^n-1 (3,7,15..)

```
RSB     Ra,Ra,Ra,LSL #n
```

Multiplication by 6

```
ADD     Ra,Ra,Ra,LSL #1     ;  Multiply by 3
MOV     Ra,Ra,LSL#1         ;  and then by 2
```

Multiply by 10 and add in extra number

```
ADD     Ra,Ra,Ra,LSL#2      ;  Multiply by 5
ADD     Ra,Rc,Ra,LSL#1      ;  Multiply by 2 and add in next digit
```

General recursive method for Rb := Ra*C, C a constant:

1. If C even, say C = 2^n*D, D odd:

```
D=1:    MOV   Rb,Ra,LSL #n
D<>1:   {Rb := Ra*D}
MOV     Rb,Rb,LSL #n
```

2. If C MOD 4 = 1, say C = 2^n*D+1, D odd, n>1:

```
D=1:    ADD   Rb,Ra,Ra,LSL #n
D<>1:   {Rb := Ra*D}
ADD     Rb,Ra,Rb,LSL #n
```

3. If C MOD 4 = 3, say C = 2^n*D-1, D odd, n>1:

```
D=1:    RSB   Rb,Ra,Ra,LSL #n
D<>1:   {Rb := Ra*D}
RSB     Rb,Ra,Rb,LSL #n
```

This is not quite optimal, but close. An example of its non-optimality is multiply by 45 which is done by:

```
RSB     Rb,Ra,Ra,LSL#2      ;  Multiply by 3
RSB     Rb,Ra,Rb,LSL#2      ;  Multiply by 4*3-1 = 11
ADD     Rb,Ra,Rb,LSL# 2     ;  Multiply by 4*11+1 = 45
```

rather than by:

```
ADD     Rb,Ra,Ra,LSL#3      ;  Multiply by 9
ADD     Rb,Rb,Rb,LSL#2      ;  Multiply by 5*9 = 45
```

LOADING A WORD FROM AN UNKNOWN ALIGNMENT

```
                                ;  Enter with address in Ra (32 bits) uses
                                ;  Rb, Rc result in Rd. Note d must be less than c e.g. 0,1
BIC       Rb,Ra,#3              ;  Get word aligned address
LDMIA     Rb,{Rd,Rc}            ;  Get 64 bits containing answer
AND       Rb,Ra,#3              ;  Correction factor in bytes
MOVS      Rb,Rb,LSL#3           ;   ...now in bits and test if aligned
MOVNE     Rd,Rd,LSR Rb          ;  Produce bottom of result word (if not aligned)
RSBNE     Rb,Rb,#32             ;  Get other shift amount
ORRNE     Rd,Rd,Rc,LSL Rb       ;  Combine two halves to get result
```

NOTES

THUMB INSTRUCTION SET FORMAT

The thumb instruction sets are 16-bit versions of ARM instruction sets (32-bit format). The ARM instructions are reduced to 16-bit versions, Thumb instructions, at the cost of versatile functions of the ARM instruction sets. The thumb instructions are decompressed to the ARM instructions by the Thumb decompressor inside the ARM7TDMI core.

As the Thumb instructions are compressed ARM instructions, the Thumb instructions have the 16-bit format instructions and have some restrictions. The restrictions by 16-bit format is fully notified for using the Thumb instructions.

FORMAT SUMMARY

The THUMB instruction set formats are shown in the following figure.

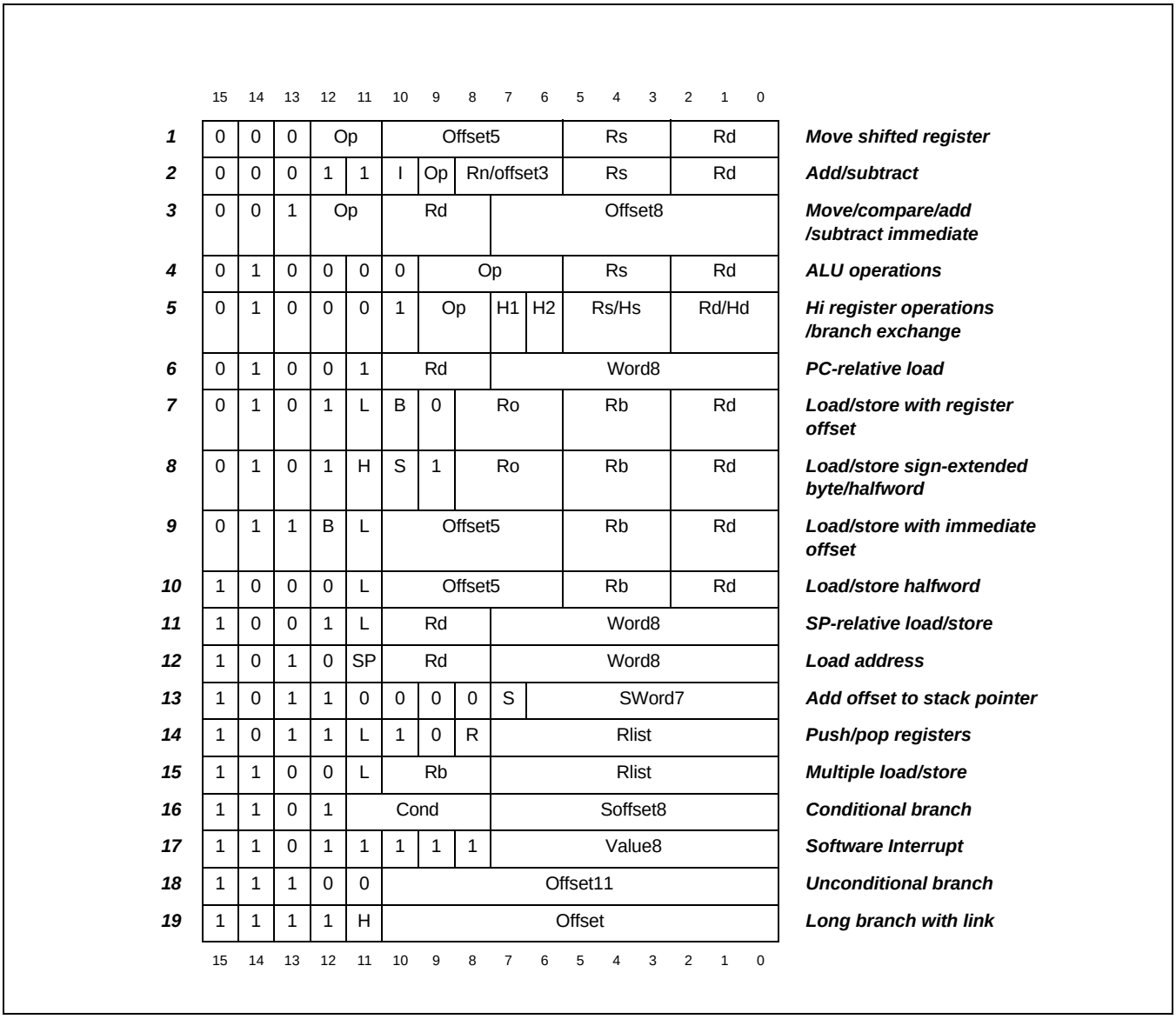

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 0 0 0 Op Offset5 Rs Rd Move shifted register
2 0 0 0 1 1 I Op Rn/offset3 Rs Rd Add/subtract
3 0 0 1 Op Rd Offset8 Move/compare/add /subtract immediate
4 0 1 0 0 0 0 Op Rs Rd ALU operations
5 0 1 0 0 0 1 Op H1 H2 Rs/Hs Rd/Hd Hi register operations /branch exchange
6 0 1 0 0 1 Rd Word8 PC-relative load
7 0 1 0 1 L B 0 Ro Rb Rd Load/store with register offset
8 0 1 0 1 H S 1 Ro Rb Rd Load/store sign-extended byte/halfword
9 0 1 1 B L Offset5 Rb Rd Load/store with immediate offset
10 1 0 0 0 L Offset5 Rb Rd Load/store halfword
11 1 0 0 1 L Rd Word8 SP-relative load/store
12 1 0 1 0 SP Rd Word8 Load address
13 1 0 1 1 0 0 0 0 S SWord7 Add offset to stack pointer
14 1 0 1 1 L 1 0 R Rlist Push/pop registers
15 1 1 0 0 L Rb Rlist Multiple load/store
16 1 1 0 1 Cond Soffset8 Conditional branch
17 1 1 0 1 1 1 1 1 Value8 Software Interrupt
18 1 1 1 0 0 Offset11 Unconditional branch
19 1 1 1 1 H Offset Long branch with link
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Figure 3-29. THUMB Instruction Set Formats

OPCODE SUMMARY

The following table summarizes the THUMB instruction set. For further information about a particular instruction please refer to the sections listed in the right-most column.

Table 3-7. THUMB Instruction Set Opcodes

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
ADC	Add with Carry	4	–	4
ADD	Add	4	4	4(1)
AND	AND	4	–	4
ASR	Arithmetic Shift Right	4	–	4
B	Unconditional branch	4	–	–
B*xx*	Conditional branch	4	–	–
BIC	Bit Clear	4	–	4
BL	Branch and Link	–	–	–
BX	Branch and Exchange	4	4	–
CMN	Compare Negative	4	–	4
CMP	Compare	4	4	4
EOR	EOR	4	–	4
LDMIA	Load multiple	4	–	–
LDR	Load word	4	–	–
LDRB	Load byte	4	–	–
LDRH	Load halfword	4	–	–
LSL	Logical Shift Left	4	–	4
LDSB	Load sign-extended byte	4	–	–
LDSH	Load sign-extended halfword	4	–	–
LSR	Logical Shift Right	4	–	4
MOV	Move register	4	4	4(2)
MUL	Multiply	4	–	4
MVN	Move Negative register	4	–	4
NEG	Negate	4	–	4
ORR	OR	4	–	4
POP	Pop registers	4	–	–

Table 3-7. THUMB Instruction Set Opcodes (Continued)

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
PUSH	Push registers	4	–	–
ROR	Rotate Right	4	–	4
SBC	Subtract with Carry	4	–	4
STMIA	Store Multiple	4	–	–
STR	Store word	4	–	–
STRB	Store byte	4	–	–
STRH	Store halfword	4	–	–
SWI	Software Interrupt	–	–	–
SUB	Subtract	4	–	4
TST	Test bits	4	–	4

NOTES

1. The condition codes are unaffected by the format 5, 12 and 13 versions of this instruction.
2. The condition codes are unaffected by the format 5 version of this instruction.

FORMAT 1: MOVE SHIFTED REGISTER

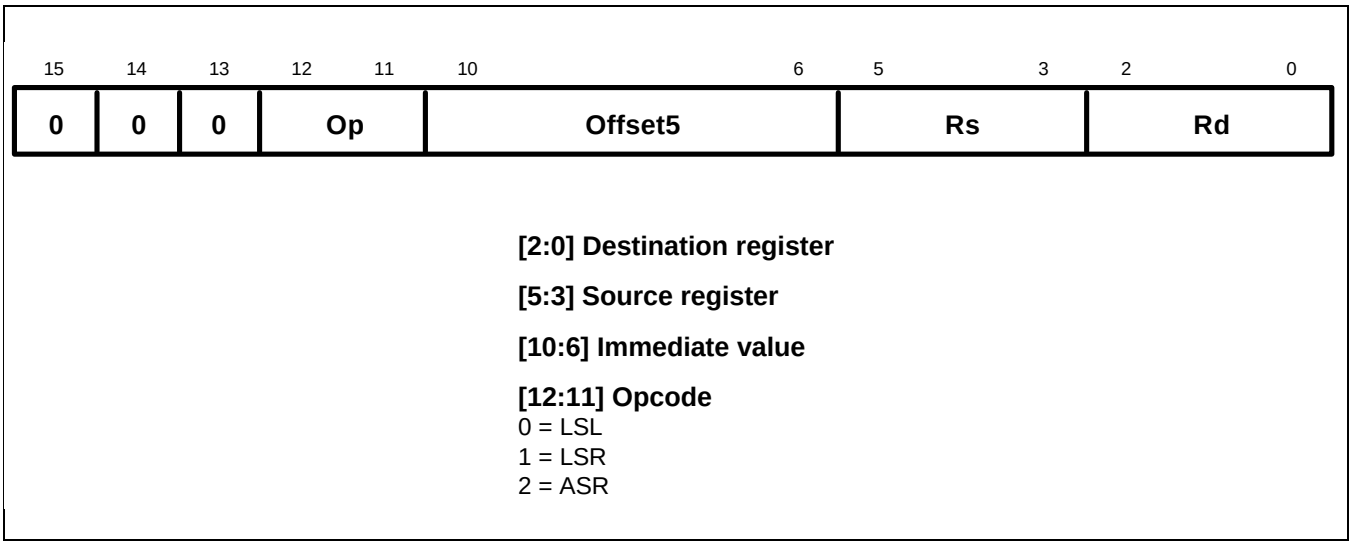

15 14 13 12 11 10 6 5 3 2 0
0 0 0 Op Offset5 Rs Rd
[2:0] Destination register
[5:3] Source register
[10:6] Immediate value
[12:11] Opcode
0 = LSL
1 = LSR
2 = ASR

Figure 3-30. Format 1

OPERATION

These instructions move a shifted value between Lo registers. The THUMB assembler syntax is shown in Table 3-8.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-8. Summary of Format 1 Instructions

OP	THUMB assembler	ARM equivalent	Action
00	LSL Rd, Rs, #Offset5	MOVS Rd, Rs, LSL #Offset5	Shift Rs left by a 5-bit immediate value and store the result in Rd.
01	LSR Rd, Rs, #Offset5	MOVS Rd, Rs, LSR #Offset5	Perform logical shift right on Rs by a 5-bit immediate value and store the result in Rd.
10	ASR Rd, Rs, #Offset5	MOVS Rd, Rs, ASR #Offset5	Perform arithmetic shift right on Rs by a 5-bit immediate value and store the result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-8. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
LSR     R2, R5, #27     ; Logical shift right the contents
                        ; of R5 by 27 and store the result in R2.
                        ; Set condition codes on the result.
```

FORMAT 2: ADD/SUBTRACT

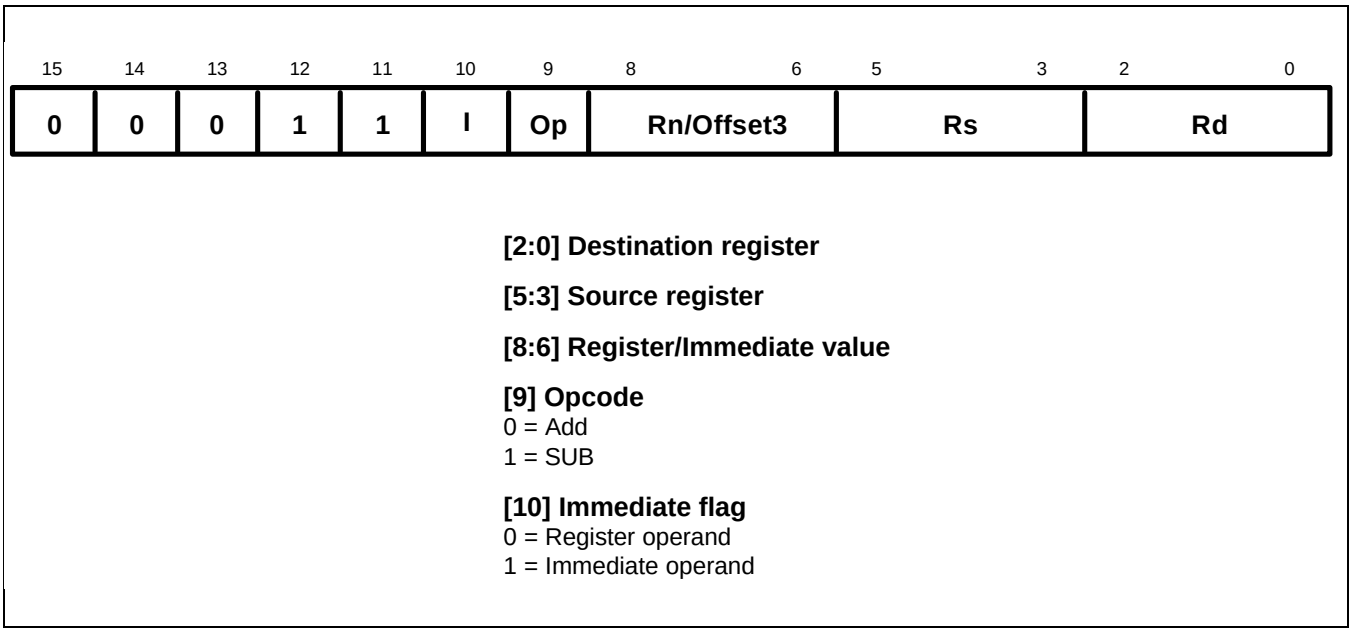

15 14 13 12 11 10 9 8 6 5 3 2 0
0 0 0 1 1 I Op Rn/Offset3 Rs Rd
[2:0] Destination register
[5:3] Source register
[8:6] Register/Immediate value
[9] Opcode
0 = Add
1 = SUB
[10] Immediate flag
0 = Register operand
1 = Immediate operand

Figure 3-31. Format 2

OPERATION

These instructions allow the contents of a Lo register or a 3-bit immediate value to be added to or subtracted from a Lo register. The THUMB assembler syntax is shown in Table 3-9.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-9. Summary of Format 2 Instructions

Op	I	THUMB assembler	ARM equivalent	Action
0	0	ADD Rd, Rs, Rn	ADDS Rd, Rs, Rn	Add contents of Rn to contents of Rs. Place result in Rd.
0	1	ADD Rd, Rs, #Offset3	ADDS Rd, Rs, #Offset3	Add 3-bit immediate value to contents of Rs. Place result in Rd.
1	0	SUB Rd, Rs, Rn	SUBS Rd, Rs, Rn	Subtract contents of Rn from contents of Rs. Place result in Rd.
1	1	SUB Rd, Rs, #Offset3	SUBS Rd, Rs, #Offset3	Subtract 3-bit immediate value from contents of Rs. Place result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-9. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
ADD     R0, R3, R4      ;  R0 := R3 + R4 and set condition codes on the result.
SUB     R6, R2, #6      ;  R6 := R2 – 6 and set condition codes.
```

FORMAT 3: MOVE/COMPARE/ADD/SUBTRACT IMMEDIATE

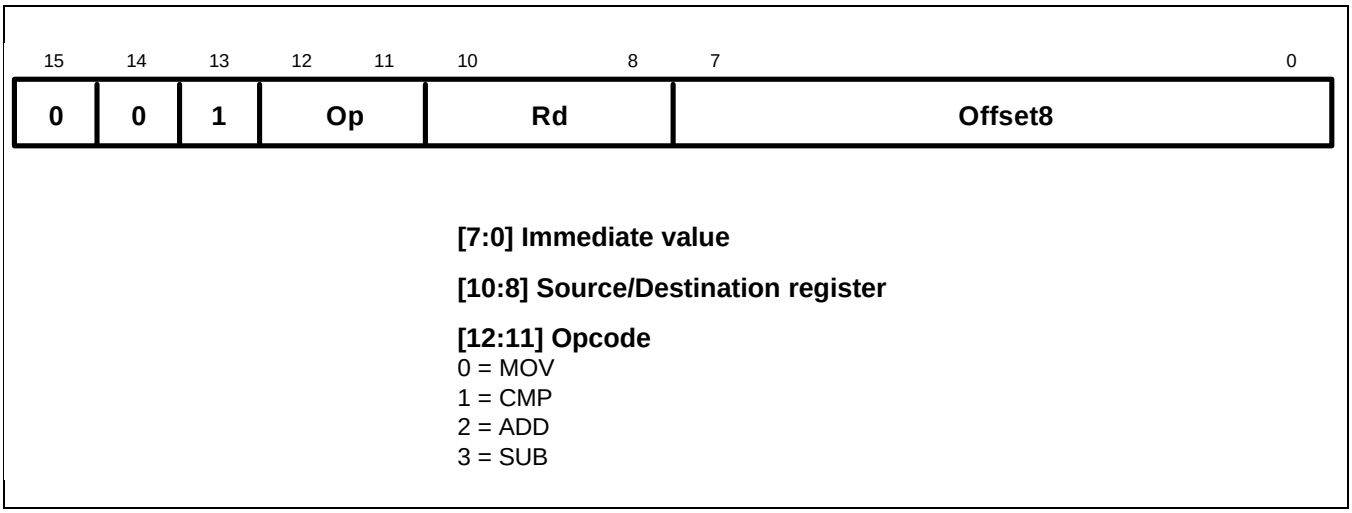


Figure 3-32. Format 3

OPERATIONS

The instructions in this group perform operations between a Lo register and an 8-bit immediate value. The THUMB assembler syntax is shown in Table 3-10.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-10. Summary of Format 3 Instructions

Op	THUMB assembler	ARM equivalent	Action
00	MOV Rd, #Offset8	MOVS Rd, #Offset8	Move 8-bit immediate value into Rd.
01	CMP Rd, #Offset8	CMP Rd, #Offset8	Compare contents of Rd with 8-bit immediate value.
10	ADD Rd, #Offset8	ADDS Rd, Rd, #Offset8	Add 8-bit immediate value to contents of Rd and place the result in Rd.
11	SUB Rd, #Offset8	SUBS Rd, Rd, #Offset8	Subtract 8-bit immediate value from contents of Rd and place the result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-10. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
MOV     R0, #128        ;   R0 := 128 and set condition codes
CMP     R2, #62         ;   Set condition codes on R2 – 62
ADD     R1, #255        ;   R1 := R1 + 255 and set condition codes
SUB     R6, #145        ;   R6 := R6 – 145 and set condition codes
```

FORMAT 4: ALU OPERATIONS

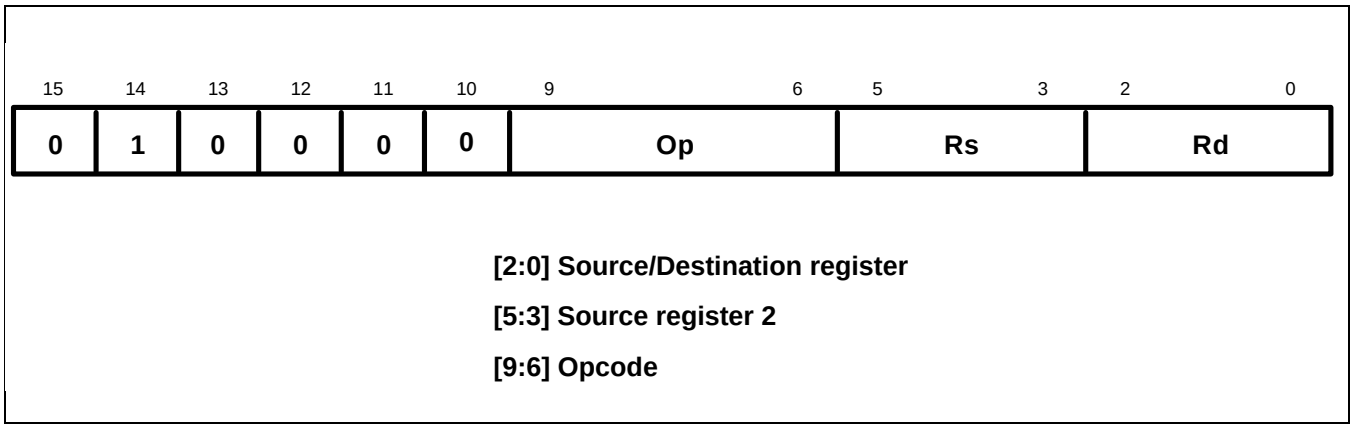


Figure 3-33. Format 4

OPERATION

The following instructions perform ALU operations on a Lo register pair.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-11. Summary of Format 4 Instructions

OP	THUMB assembler	ARM equivalent	Action
0000	AND Rd, Rs	ANDS Rd, Rd, Rs	Rd:= Rd AND Rs
0001	EOR Rd, Rs	EORS Rd, Rd, Rs	Rd:= Rd EOR Rs
0010	LSL Rd, Rs	MOVS Rd, Rd, LSL Rs	Rd := Rd << Rs
0011	LSR Rd, Rs	MOVS Rd, Rd, LSR Rs	Rd := Rd >> Rs
0100	ASR Rd, Rs	MOVS Rd, Rd, ASR Rs	Rd := Rd ASR Rs
0101	ADC Rd, Rs	ADCS Rd, Rd, Rs	Rd := Rd + Rs + C-bit
0110	SBC Rd, Rs	SBCS Rd, Rd, Rs	Rd := Rd – Rs – NOT C-bit
0111	ROR Rd, Rs	MOVS Rd, Rd, ROR Rs	Rd := Rd ROR Rs
1000	TST Rd, Rs	TST Rd, Rs	Set condition codes on Rd AND Rs
1001	NEG Rd, Rs	RSBS Rd, Rs, #0	Rd = – Rs
1010	CMP Rd, Rs	CMP Rd, Rs	Set condition codes on Rd – Rs
1011	CMN Rd, Rs	CMN Rd, Rs	Set condition codes on Rd + Rs
1100	ORR Rd, Rs	ORRS Rd, Rd, Rs	Rd := Rd OR Rs
1101	MUL Rd, Rs	MULS Rd, Rs, Rd	Rd := Rs * Rd
1110	BIC Rd, Rs	BICS Rd, Rd, Rs	Rd := Rd AND NOT Rs
1111	MVN Rd, Rs	MVNS Rd, Rs	Rd := NOT Rs

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-11. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
EOR     R3, R4      ; R3 := R3 EOR R4 and set condition codes
ROR     R1, R0      ; Rotate Right R1 by the value in R0, store
                    ; the result in R1 and set condition codes
NEG     R5, R3      ; Subtract the contents of R3 from zero,
                    ; store the result in R5. Set condition codes ie R5 = – R3
CMP     R2, R6      ; Set the condition codes on the result of R2 – R6
MUL     R0, R7      ; R0 := R7 * R0 and set condition codes
```

FORMAT 5: HI-REGISTER OPERATIONS/BRANCH EXCHANGE

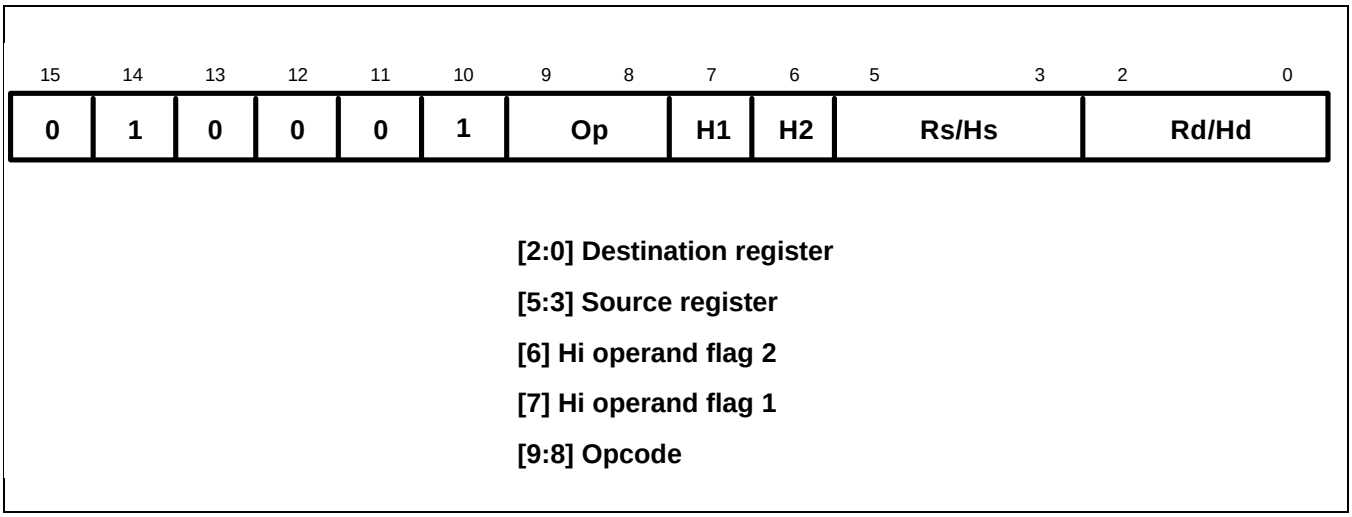

15 14 13 12 11 10 9 8 7 6 5 3 2 0
0 1 0 0 0 1 Op H1 H2 Rs/Hs Rd/Hd
[2:0] Destination register
[5:3] Source register
[6] Hi operand flag 2
[7] Hi operand flag 1
[9:8] Opcode

Figure 3-34. Format 5

OPERATION

There are four sets of instructions in this group. The first three allow ADD, CMP and MOV operations to be performed between Lo and Hi registers, or a pair of Hi registers. The fourth, BX, allows a Branch to be performed which may also be used to switch processor state. The THUMB assembler syntax is shown in Table 3-12.

NOTE

In this group only CMP (Op = 01) sets the CPSR condition codes.

The action of H1= 0, H2 = 0 for Op = 00 (ADD), Op =01 (CMP) and Op = 10 (MOV) is undefined, and should not be used.

Table 3-12. Summary of Format 5 Instructions

Op	H1	H2	THUMB assembler	ARM equivalent	Action
00	0	1	ADD Rd, Hs	ADD Rd, Rd, Hs	Add a register in the range 8-15 to a register in the range 0-7.
00	1	0	ADD Hd, Rs	ADD Hd, Hd, Rs	Add a register in the range 0-7 to a register in the range 8-15.
00	1	1	ADD Hd, Hs	ADD Hd, Hd, Hs	Add two registers in the range 8-15
01	0	1	CMP Rd, Hs	CMP Rd, Hs	Compare a register in the range 0-7 with a register in the range 8-15. Set the condition code flags on the result.
01	1	0	CMP Hd, Rs	CMP Hd, Rs	Compare a register in the range 8-15 with a register in the range 0-7. Set the condition code flags on the result.

Table 3-12. Summary of Format 5 Instructions (Continued)

Op	H1	H2	THUMB assembler	ARM equivalent	Action
01	1	1	CMP Hd, Hs	CMP Hd, Hs	Compare two registers in the range 8-15. Set the condition code flags on the result.
10	0	1	MOV Rd, Hs	MOV Rd, Hs	Move a value from a register in the range 8-15 to a register in the range 0-7.
10	1	0	MOV Hd, Rs	MOV Hd, Rs	Move a value from a register in the range 0-7 to a register in the range 8-15.
10	1	1	MOV Hd, Hs	MOV Hd, Hs	Move a value between two registers in the range 8-15.
11	0	0	BX Rs	BX Rs	Perform branch (plus optional state change) to address in a register in the range 0-7.
11	0	1	BX Hs	BX Hs	Perform branch (plus optional state change) to address in a register in the range 8-15.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-12. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

THE BX INSTRUCTION

BX performs a Branch to a routine whose start address is specified in a Lo or Hi register.

Bit 0 of the address determines the processor state on entry to the routine:

Bit 0 = 0 Causes the processor to enter ARM state.
Bit 0 = 1 Causes the processor to enter THUMB state.

NOTE

The action of H1 = 1 for this instruction is undefined, and should not be used.

EXAMPLES

Hi-Register Operations

```
        ADD     PC, R5          ;  PC := PC + R5 but don't set the condition codes.
        CMP     R4, R12         ;  Set the condition codes on the result of R4 - R12.
        MOV     R15, R14        ;  Move R14 (LR) into R15 (PC)
                                ;  but don't set the condition codes,
                                ;  eg. return from subroutine.
```

Branch and Exchange

```
                                ;  Switch from THUMB to ARM state.
        ADR     R1,outofTHUMB   ;  Load address of outofTHUMB into R1.
        MOV     R11,R1
        BX      R11                 ; Transfer the contents of R11 into the PC.
                                    ; Bit 0 of R11 determines whether
                                    ; ARM or THUMB state is entered, ie. ARM state here.
        ...
        ALIGN
        CODE32
        outofTHUMB
                                ;  Now processing ARM instructions...
```

USING R15 AS AN OPERAND

If R15 is used as an operand, the value will be the address of the instruction + 4 with bit 0 cleared. Executing a BX PC in THUMB state from a non-word aligned address will result in unpredictable execution.

FORMAT 6: PC-RELATIVE LOAD

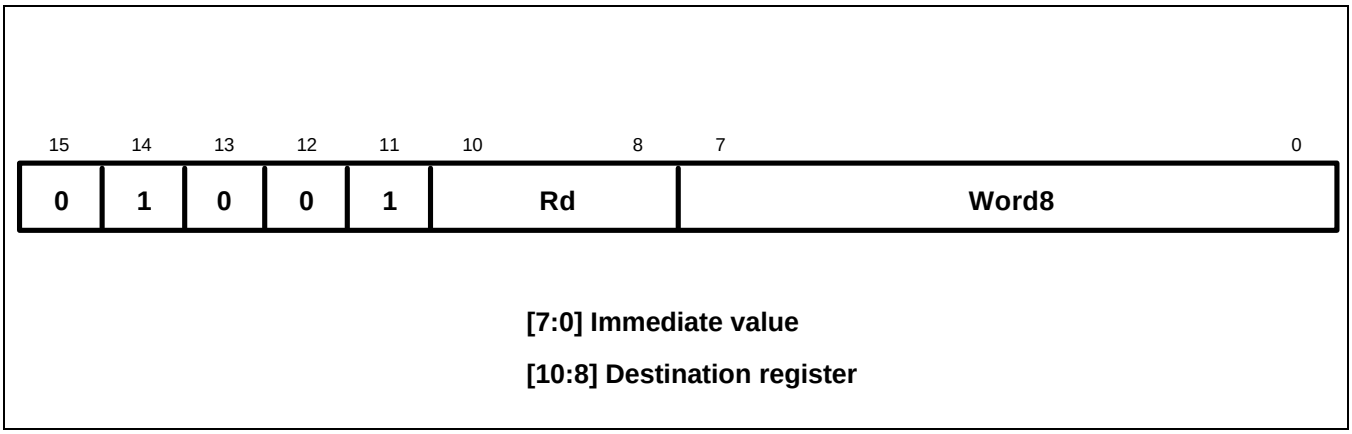


Figure 3-35. Format 6

OPERATION

This instruction loads a word from an address specified as a 10-bit immediate offset from the PC. The THUMB assembler syntax is shown below.

Table 3-13. Summary of PC-Relative Load Instruction

THUMB assembler	ARM equivalent	Action
LDR Rd, [PC, #Imm]	LDR Rd, [R15, #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the PC. Load the word from the resulting address into Rd.

NOTE: The value specified by #Imm is a full 10-bit address, but must always be word-aligned (ie with bits 1:0 set to 0), since the assembler places #Imm >> 2 in field Word 8. The value of the PC will be 4 bytes greater than the address of this instruction, but bit 1 of the PC is forced to 0 to ensure it is word aligned.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
LDR R3,[PC,#844]    ; Load into R3 the word found at the
                    ; address formed by adding 844 to PC.
                    ; bit[1] of PC is forced to zero.
                    ; Note that the THUMB opcode will contain
                    ; 211 as the Word8 value.
```

FORMAT 7: LOAD/STORE WITH REGISTER OFFSET

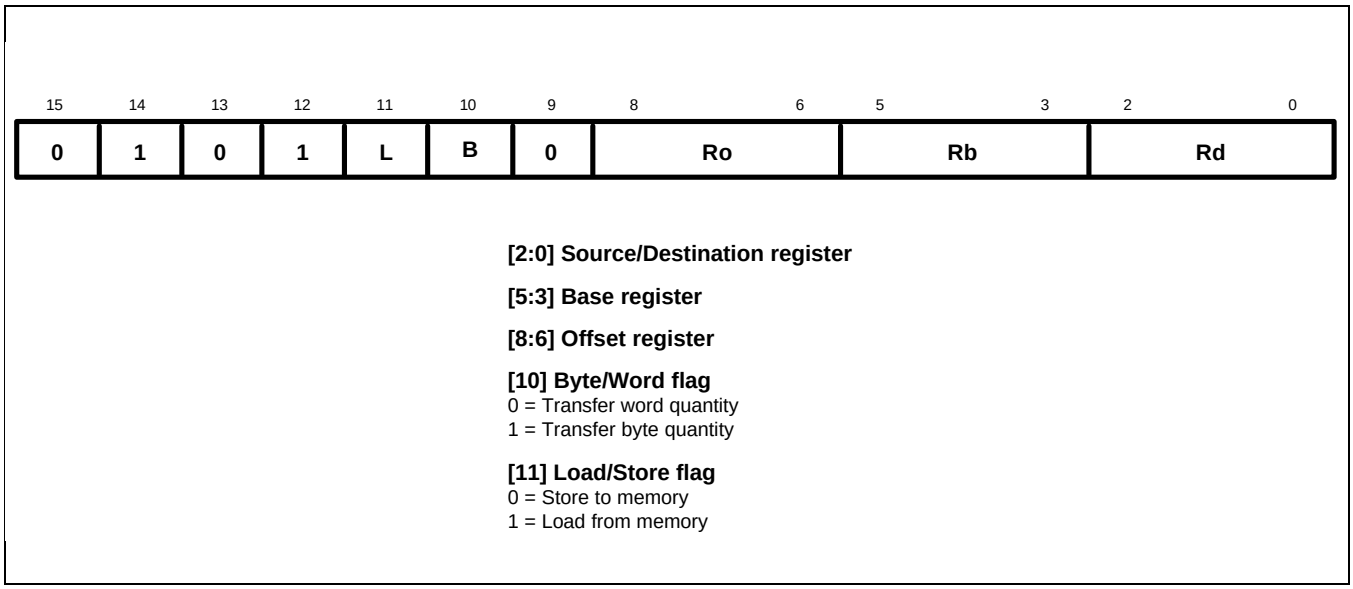

15 14 13 12 11 10 9 8 6 5 3 2 0
0 1 0 1 L B 0 Ro Rb Rd
[2:0] Source/Destination register
[5:3] Base register
[8:6] Offset register
[10] Byte/Word flag
0 = Transfer word quantity
1 = Transfer byte quantity
[11] Load/Store flag
0 = Store to memory
1 = Load from memory

Figure 3-36. Format 7

OPERATION

These instructions transfer byte or word values between registers and memory. Memory addresses are pre-indexed using an offset register in the range 0-7. The THUMB assembler syntax is shown in Table 3-14.

Table 3-14. Summary of Format 7 Instructions

L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, Ro]	STR Rd, [Rb, Ro]	Pre-indexed word store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the contents of Rd at the address.
0	1	STRB Rd, [Rb, Ro]	STRB Rd, [Rb, Ro]	Pre-indexed byte store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the byte value in Rd at the resulting address.
1	0	LDR Rd, [Rb, Ro]	LDR Rd, [Rb, Ro]	Pre-indexed word load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the contents of the address into Rd.

Table 3-14. Summary of Format 7 Instructions (Continued)

L	B	THUMB assembler	ARM equivalent	Action
1	1	LDRB Rd, [Rb, Ro]	LDRB Rd, [Rb, Ro]	Pre-indexed byte load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the byte value at the resulting address.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-14. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STR     R3, [R2,R6]     ; Store word in R3 at the address
                        ; formed by adding R6 to R2.
LDRB    R2, [R0,R7]     ; Load into R2 the byte found at
                        ; the address formed by adding R7 to R0.
```

FORMAT 8: LOAD/STORE SIGN-EXTENDED BYTE/HALFWORD

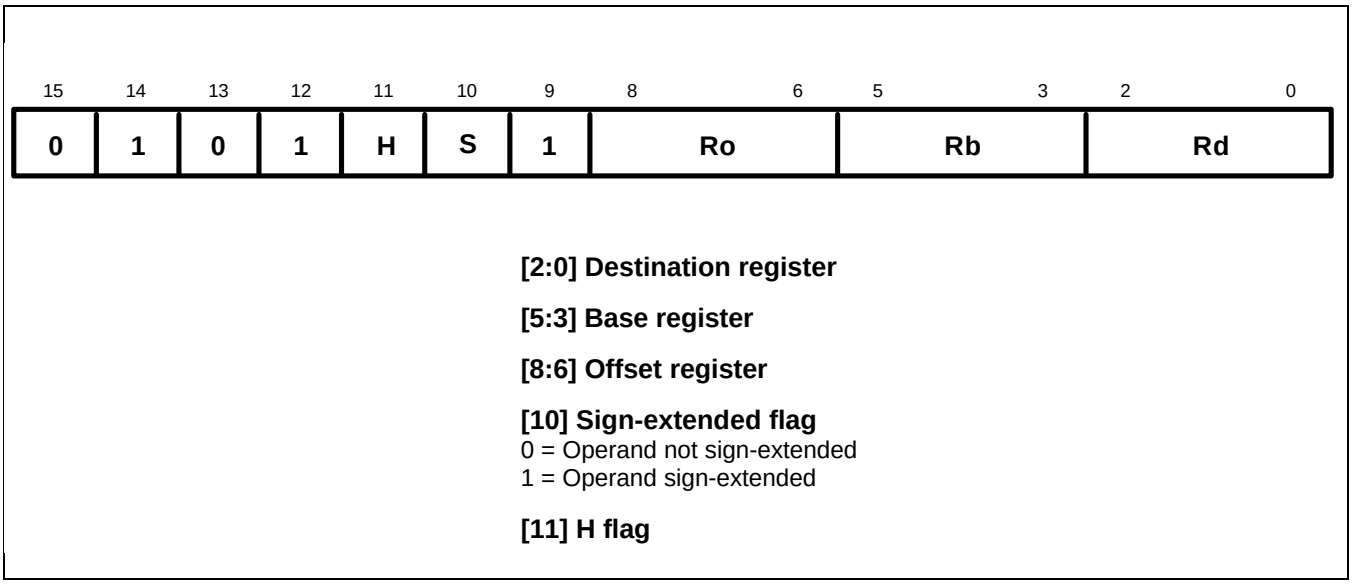

15 14 13 12 11 10 9 8 6 5 3 2 0
0 1 0 1 H S 1 Ro Rb Rd
[2:0] Destination register
[5:3] Base register
[8:6] Offset register
[10] Sign-extended flag
0 = Operand not sign-extended
1 = Operand sign-extended
[11] H flag

Figure 3-37. Format 8

OPERATION

These instructions load optionally sign-extended bytes or halfwords, and store halfwords. The THUMB assembler syntax is shown below.

Table 3-15. Summary of format 8 instructions

S	H	THUMB assembler	ARM equivalent	Action
0	0	STRH Rd, [Rb, Ro]	STRH Rd, [Rb, Ro]	Store halfword: Add Ro to base address in Rb. Store bits 0-15 of Rd at the resulting address.
0	1	LDRH Rd, [Rb, Ro]	LDRH Rd, [Rb, Ro]	Load halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to 0.
1	0	LDSB Rd, [Rb, Ro]	LDRSB Rd, [Rb, Ro]	Load sign-extended byte: Add Ro to base address in Rb. Load bits 0-7 of Rd from the resulting address, and set bits 8-31 of Rd to bit 7.
1	1	LDSH Rd, [Rb, Ro]	LDRSH Rd, [Rb, Ro]	Load sign-extended halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to bit 15.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-15. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STRH    R4, [R3, R0]    ;  Store the lower 16 bits of R4 at the
                        ;  address formed by adding R0 to R3.
LDSB    R2, [R7, R1]    ;  Load into R2 the sign extended byte
                        ;  found at the address formed by adding R1 to R7.
LDSH    R3, [R4, R2]    ;  Load into R3 the sign extended halfword
                        ;  found at the address formed by adding R2 to R4.
```

FORMAT 9: LOAD/STORE WITH IMMEDIATE OFFSET

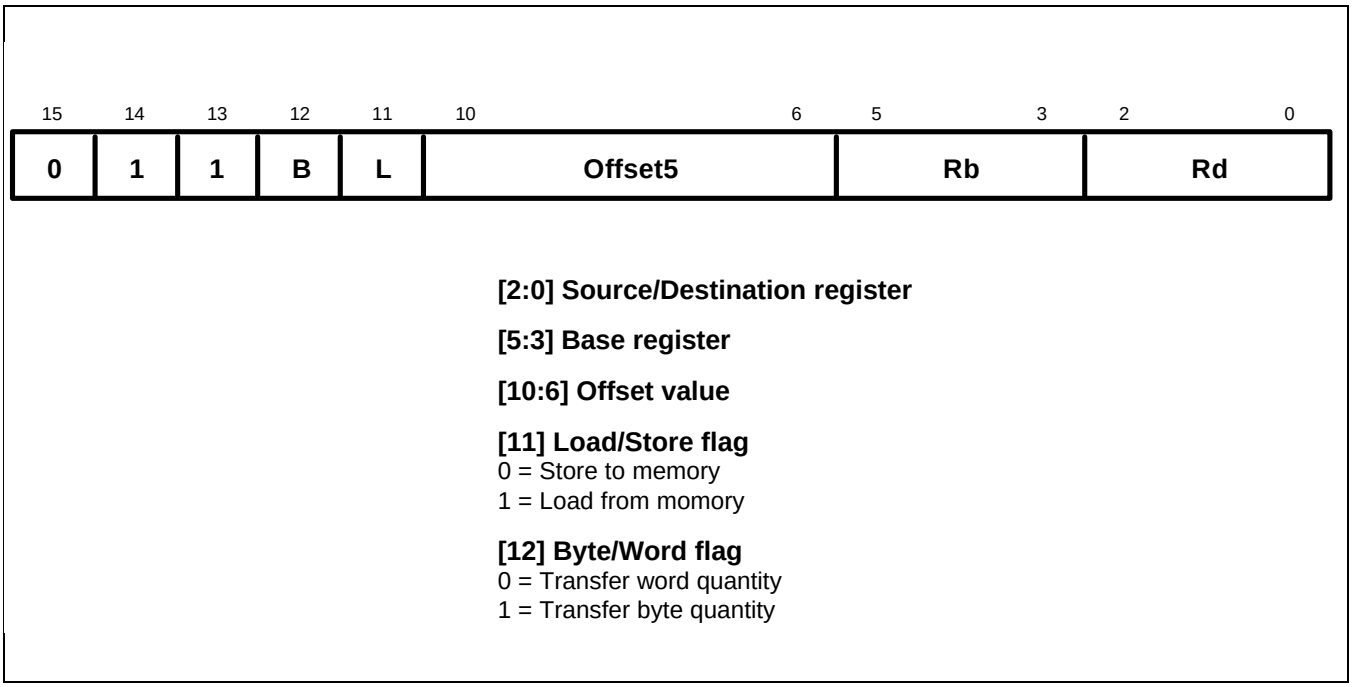


Figure 3-38. Format 9

OPERATION

These instructions transfer byte or word values between registers and memory using an immediate 5 or 7-bit offset. The THUMB assembler syntax is shown in Table 3-16.

Table 3-16. Summary of Format 9 Instructions

L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, #Imm]	STR Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the contents of Rd at the address.
1	0	LDR Rd, [Rb, #Imm]	LDR Rd, [Rb, #Imm]	Calculate the source address by adding together the value in Rb and Imm. Load Rd from the address.
0	1	STRB Rd, [Rb, #Imm]	STRB Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the byte value in Rd at the address.
1	1	LDRB Rd, [Rb, #Imm]	LDRB Rd, [Rb, #Imm]	Calculate source address by adding together the value in Rb and Imm. Load the byte value at the address into Rd.

NOTE: For word accesses (B = 0), the value specified by #Imm is a full 7-bit address, but must be word-aligned (ie with bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Offset5 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-16. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
LDR     R2, [R5,#116]   ; Load into R2 the word found at the
                        ; address formed by adding 116 to R5.
                        ; Note that the THUMB opcode will
                        ; contain 29 as the Offset5 value.
STRB    R1, [R0,#13]    ; Store the lower 8 bits of R1 at the
                        ; address formed by adding 13 to R0.
                        ; Note that the THUMB opcode will
                        ; contain 13 as the Offset5 value.
```

FORMAT 10: LOAD/STORE HALFWORD

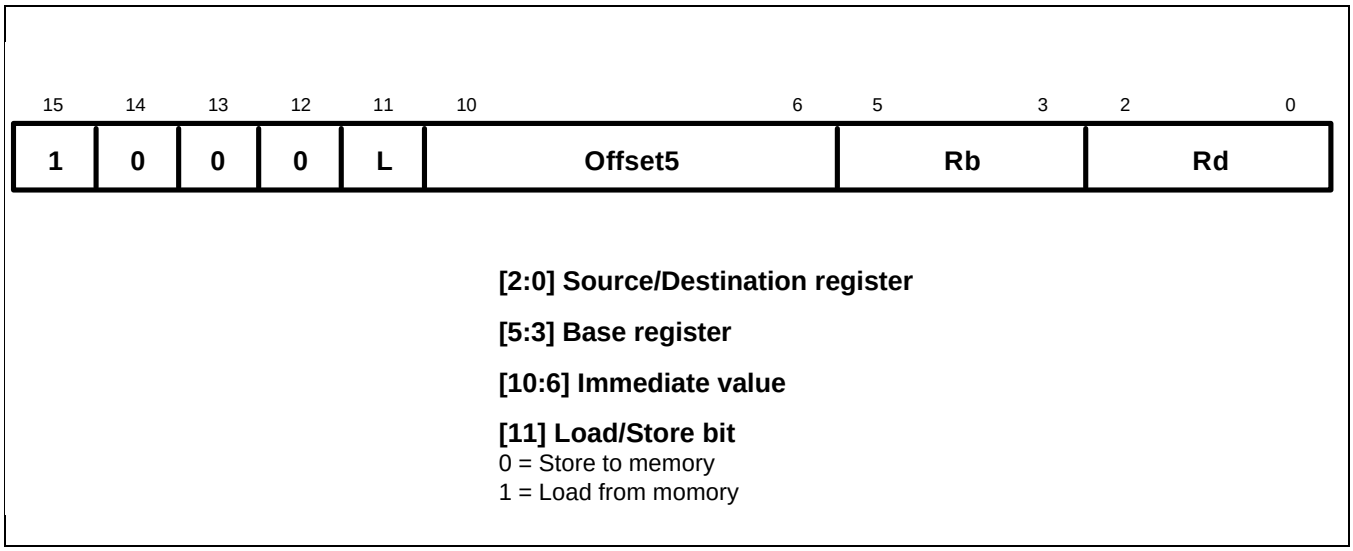

15 14 13 12 11 10 6 5 3 2 0
1 0 0 0 L Offset5 Rb Rd
[2:0] Source/Destination register
[5:3] Base register
[10:6] Immediate value
[11] Load/Store bit
0 = Store to memory
1 = Load from momory

Figure 3-39. Format 10

OPERATION

These instructions transfer halfword values between a Lo register and memory. Addresses are pre-indexed, using a 6-bit immediate value. The THUMB assembler syntax is shown in Table 3-17.

Table 3-17. Halfword Data Transfer Instructions

L	THUMB assembler	ARM equivalent	Action
0	STRH Rd, [Rb, #Imm]	STRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb and store bits 0–15 of Rd at the resulting address.
1	LDRH Rd, [Rb, #Imm]	LDRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb. Load bits 0-15 from the resulting address into Rd and set bits 16-31 to zero.

NOTE: #Imm is a full 6-bit address but must be halfword-aligned (ie with bit 0 set to 0) since the assembler places #Imm >> 1 in the Offset5 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-17. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
        STRH    R6, [R1, #56]   ;  Store the lower 16 bits of R4 at the address formed by
                                ;  adding 56 R1. Note that the THUMB opcode will
contain
                                ;  28 as the Offset5 value.
        LDRH    R4, [R7, #4]    ;  Load into R4 the halfword found at the address formed
by
                                ;  adding 4 to R7. Note that the THUMB opcode will
contain
                                ;  2 as the Offset5 value.
```

FORMAT 11: SP-RELATIVE LOAD/STORE

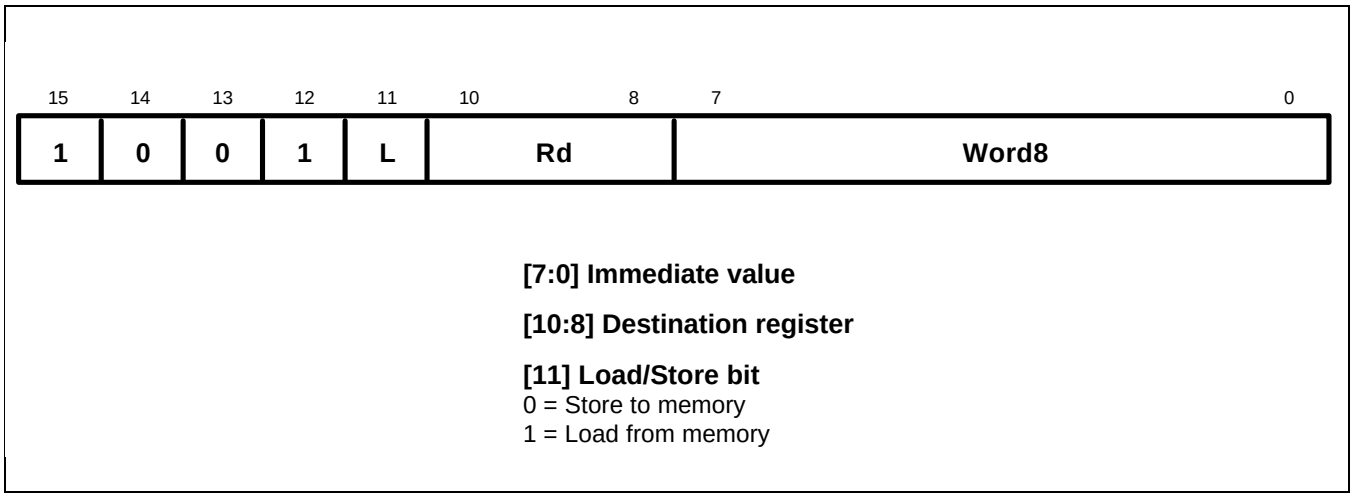

15 14 13 12 11 10 8 7 0
1 0 0 1 L Rd Word8
[7:0] Immediate value
[10:8] Destination register
[11] Load/Store bit
0 = Store to memory
1 = Load from memory

Figure 3-40. Format 11

OPERATION

The instructions in this group perform an SP-relative load or store.The THUMB assembler syntax is shown in the following table.

Table 3-18. SP-Relative Load/Store Instructions

L	THUMB assembler	ARM equivalent	Action
0	STR Rd, [SP, #Imm]	STR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Store the contents of Rd at the resulting address.
1	LDR Rd, [SP, #Imm]	LDR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Load the word from the resulting address into Rd.

NOTE: The offset supplied in #Imm is a full 10-bit address, but must always be word-aligned (ie bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Word8 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-18. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STR     R4, [SP,#492]   ; Store the contents of R4 at the address
                        ; formed by adding 492 to SP (R13).
                        ; Note that the THUMB opcode will contain
                        ; 123 as the Word8 value.
```

FORMAT 12: LOAD ADDRESS

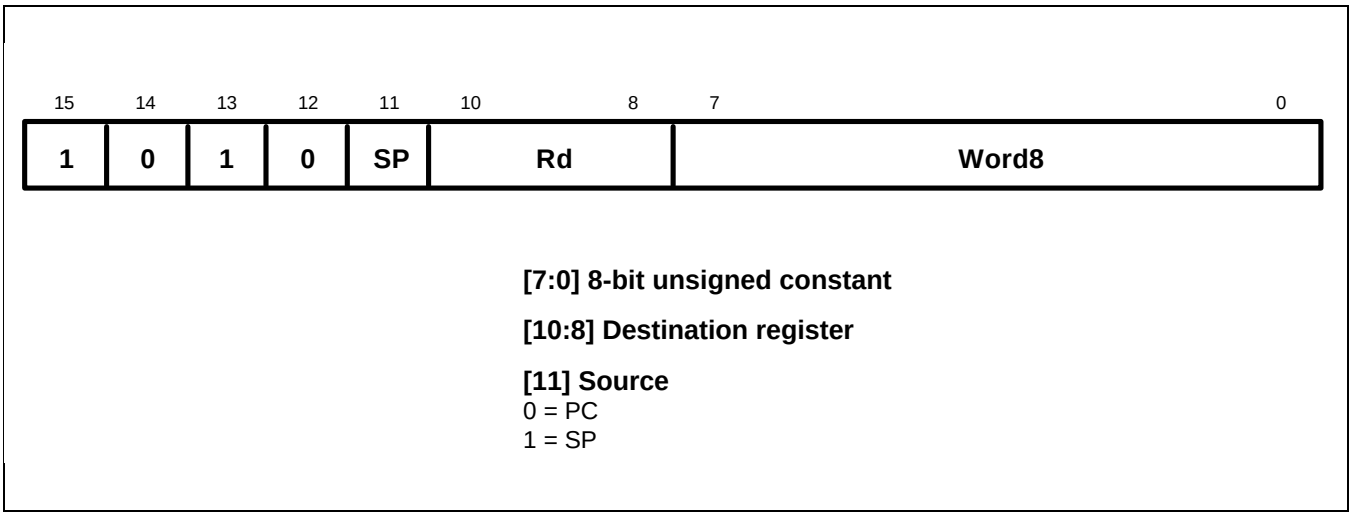


Figure 3-41. Format 12

OPERATION

These instructions calculate an address by adding an 10-bit constant to either the PC or the SP, and load the resulting address into a register. The THUMB assembler syntax is shown in the following table.

Table 3-19. Load Address

SP	THUMB assembler	ARM equivalent	Action
0	ADD Rd, PC, #Imm	ADD Rd, R15, #Imm	Add #Imm to the current value of the program counter (PC) and load the result into Rd.
1	ADD Rd, SP, #Imm	ADD Rd, R13, #Imm	Add #Imm to the current value of the stack pointer (SP) and load the result into Rd.

NOTE: The value specified by #Imm is a full 10-bit value, but this must be word-aligned (ie with bits 1:0 set to 0) since the assembler places #Imm >> 2 in field Word 8.

Where the PC is used as the source register (SP = 0), bit 1 of the PC is always read as 0. The value of the PC will be 4 bytes greater than the address of the instruction before bit 1 is forced to 0.

The CPSR condition codes are unaffected by these instructions.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-19. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
ADD     R2, PC, #572    ; R2 := PC + 572, but don't set the
                        ; condition codes. bit[1] of PC is forced to zero.
                        ; Note that the THUMB opcode will
                        ; contain 143 as the Word8 value.
ADD     R6, SP, #212    ; R6 := SP (R13) + 212, but don't
                        ; set the condition codes.
                        ; Note that the THUMB opcode will
                        ; contain 53 as the Word 8 value.
```

FORMAT 13: ADD OFFSET TO STACK POINTER

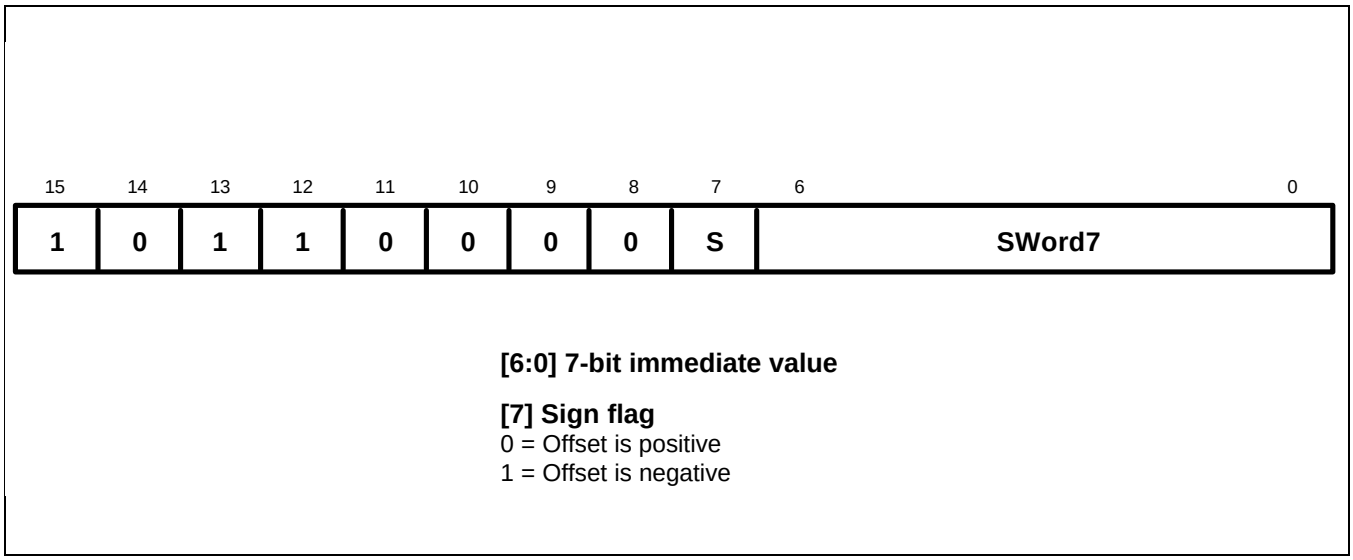


Figure 3-42. Format 13

OPERATION

This instruction adds a 9-bit signed constant to the stack pointer. The following table shows the THUMB assembler syntax.

Table 3-20. The ADD SP Instruction

S	THUMB assembler	ARM equivalent	Action
0	ADD SP, #Imm	ADD R13, R13, #Imm	Add #Imm to the stack pointer (SP).
1	ADD SP, #-Imm	SUB R13, R13, #Imm	Add #-Imm to the stack pointer (SP).

NOTE: The offset specified by #Imm can be up to -/+ 508, but must be word-aligned (ie with bits 1:0 set to 0) since the assembler converts #Imm to an 8-bit sign + magnitude number before placing it in field SWord7. The condition codes are not set by this instruction.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-20. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
        ADD     SP, #268        ;  SP (R13) := SP + 268, but don't set the condition
codes.
                                ;  Note that the THUMB opcode will
                                ;  contain 67 as the Word7 value and S=0.
        ADD     SP, #-104       ;  SP (R13) := SP - 104, but don't set the condition codes.
                                ;  Note that the THUMB opcode will contain
                                ;  26 as the Word7 value and S=1.
```

FORMAT 14: PUSH/POP REGISTERS

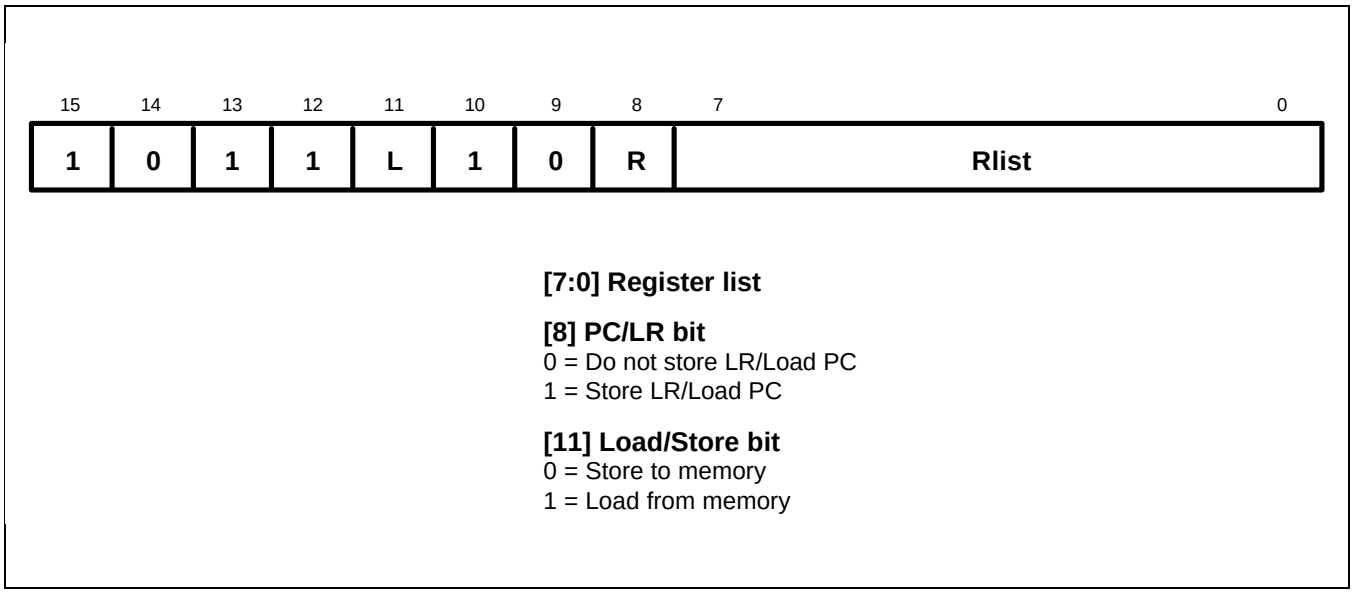

15 14 13 12 11 10 9 8 7 0
1 0 1 1 L 1 0 R Rlist
[7:0] Register list
[8] PC/LR bit
0 = Do not store LR/Load PC
1 = Store LR/Load PC
[11] Load/Store bit
0 = Store to memory
1 = Load from memory

Figure 3-43. Format 14

OPERATION

The instructions in this group allow registers 0-7 and optionally LR to be pushed onto the stack, and registers 0-7 and optionally PC to be popped off the stack. The THUMB assembler syntax is shown in Table 3-21.

NOTE

The stack is always assumed to be Full Descending.

Table 3-21. PUSH and POP Instructions

L	R	THUMB assembler	ARM equivalent	Action
0	0	PUSH { Rlist }	STMDB R13!, { Rlist }	Push the registers specified by Rlist onto the stack. Update the stack pointer.
0	1	PUSH { Rlist, LR }	STMDB R13!, { Rlist, R14 }	Push the Link Register and the registers specified by Rlist (if any) onto the stack. Update the stack pointer.
1	0	POP { Rlist }	LDMIA R13!, { Rlist }	Pop values off the stack into the registers specified by Rlist. Update the stack pointer.
1	1	POP { Rlist, PC }	LDMIA R13!, {Rlist, R15}	Pop values off the stack and load into the registers specified by Rlist. Pop the PC off the stack. Update the stack pointer.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-21. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
        PUSH    {R0-R4,LR}      ;  Store R0,R1,R2,R3,R4 and R14 (LR) at
                                ;  the stack pointed to by R13 (SP) and update R13.
                                ;  Useful at start of a sub-routine to
                                ;  save workspace and return address.
        POP     {R2,R6,PC}      ;  Load R2,R6 and R15 (PC) from the stack
                                ;  pointed to by R13 (SP) and update R13.
                                ;  Useful to restore workspace and return from sub-
routine.
```

FORMAT 15: MULTIPLE LOAD/STORE

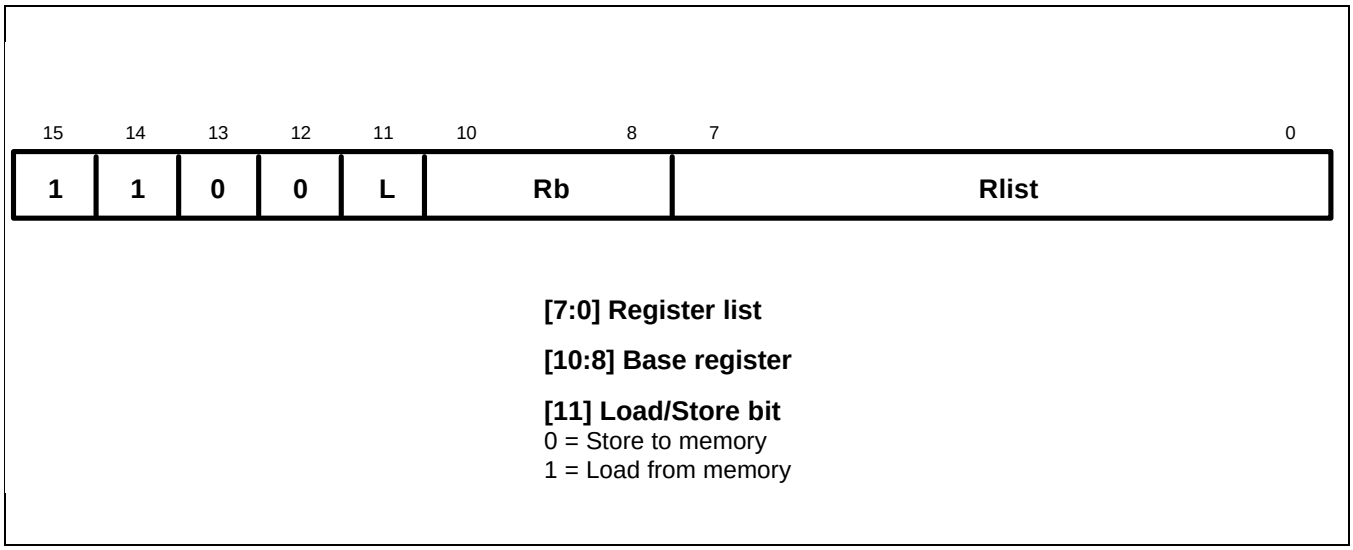


Figure 3-44. Format 15

OPERATION

These instructions allow multiple loading and storing of Lo registers. The THUMB assembler syntax is shown in the following table.

Table 3-22. The Multiple Load/Store Instructions

L	THUMB assembler	ARM equivalent	Action
0	STMIA Rb!, { Rlist }	STMIA Rb!, { Rlist }	Store the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.
1	LDMIA Rb!, { Rlist }	LDMIA Rb!, { Rlist }	Load the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-22. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
STMIA    R0!, {R3-R7}    ; Store the contents of registers R3-R7
                         ; starting at the address specified in
                         ; R0, incrementing the addresses for each word.
                         ; Write back the updated value of R0.
```

FORMAT 16: CONDITIONAL BRANCH

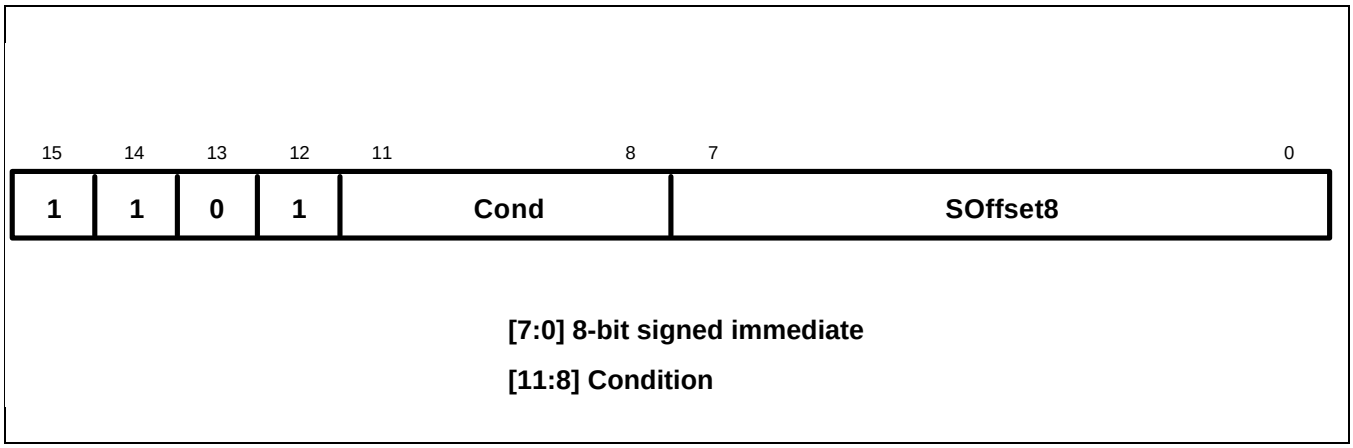


Figure 3-45. Format 16

OPERATION

The instructions in this group all perform a conditional Branch depending on the state of the CPSR condition codes. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

The THUMB assembler syntax is shown in the following table.

Table 3-23. The Conditional Branch Instructions

Cond	THUMB assembler	ARM equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)
1010	BGE label	BGE label	Branch if N set and V set, or N clear and V clear (greater or equal)

Table 3-23. The Conditional Branch Instructions (Continued)

Cond	THUMB assembler	ARM equivalent	Action
1011	BLT label	BLT label	Branch if N set and V clear, or N clear and V set (less than)
1100	BGT label	BGT label	Branch if Z clear, and either N set and V set or N clear and V clear (greater than)
1101	BLE label	BLE label	Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)

NOTES

1. While label specifies a full 9-bit two's complement address, this must always be halfword-aligned (ie with bit 0 set to 0) since the assembler actually places label >> 1 in field SOffset8.
2. Cond = 1110 is undefined, and should not be used.
   Cond = 1111 creates the SWI instruction: see .

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-23. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
        CMP R0, #45          ;  Branch to 'over' if R0 > 45.
        BGT over             ;  Note that the THUMB opcode will contain
        ...                  ;  the number of halfwords to offset.
        ...
over    ...                  ; Must be halfword aligned.
        ...
```

FORMAT 17: SOFTWARE INTERRUPT

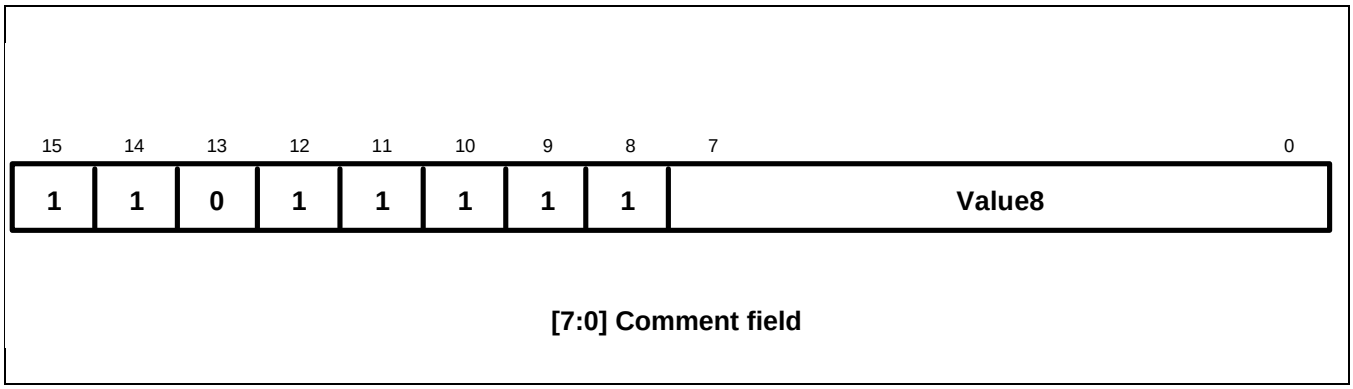


Figure 3-46. Format 17

OPERATION

The SWI instruction performs a software interrupt. On taking the SWI, the processor switches into ARM state and enters Supervisor (SVC) mode.

The THUMB assembler syntax for this instruction is shown below.

Table 3-24. The SWI Instruction

THUMB Assembler	ARM Equivalent	Action
SWI Value 8	SWI Value 8	Perform Software Interrupt: Move the address of the next instruction into LR, move CPSR to SPSR, load the SWI vector address (0x8) into the PC. Switch to ARM state and enter SVC mode.

NOTE: Value8 is used solely by the SWI handler; it is ignored by the processor.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-24. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

EXAMPLES

```
SWI 18          ; Take the software interrupt exception.
                ; Enter Supervisor mode with 18 as the
                ; requested SWI number.
```

FORMAT 18: UNCONDITIONAL BRANCH

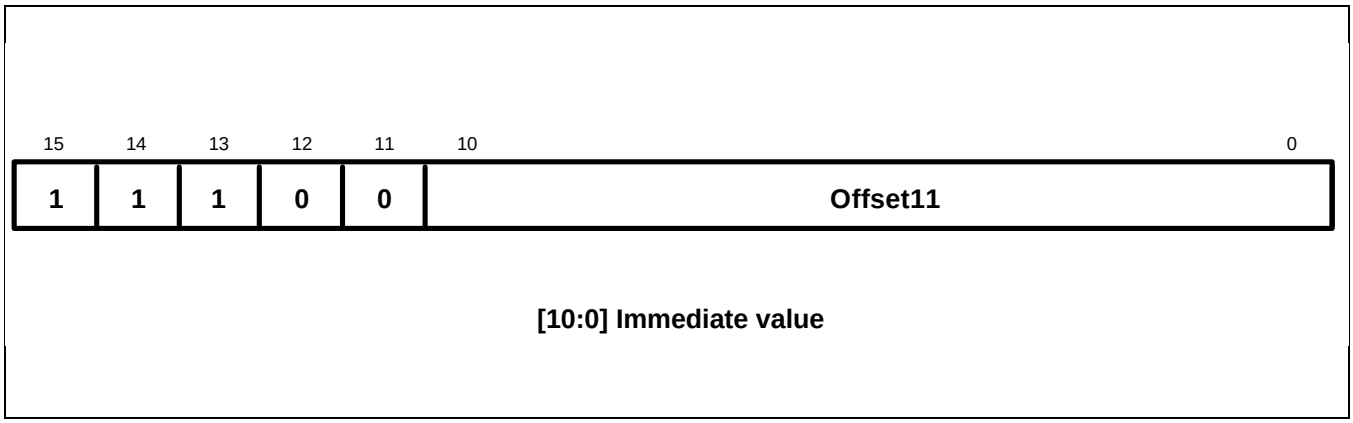


Figure 3-47. Format 18

OPERATION

This instruction performs a PC-relative Branch. The THUMB assembler syntax is shown below. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

Table 3-25. Summary of Branch Instruction

THUMB Assembler	ARM Equivalent	Action
B label	BAL label (halfword offset)	Branch PC relative +/- Offset11 << 1, where label is PC +/- 2048 bytes.

NOTE: The address specified by label is a full 12-bit two's complement address, but must always be halfword aligned (ie bit 0 set to 0), since the assembler places label >> 1 in the Offset11 field.

EXAMPLES

```
here    B here          ; Branch onto itself. Assembles to 0xE7FE.
                        ; (Note effect of PC offset).
        B jimmy         ; Branch to 'jimmy'.
        ...             ; Note that the THUMB opcode will contain the number
of
                        ; halfwords to offset.
jimmy   ...             ; Must be halfword aligned.
```

FORMAT 19: LONG BRANCH WITH LINK

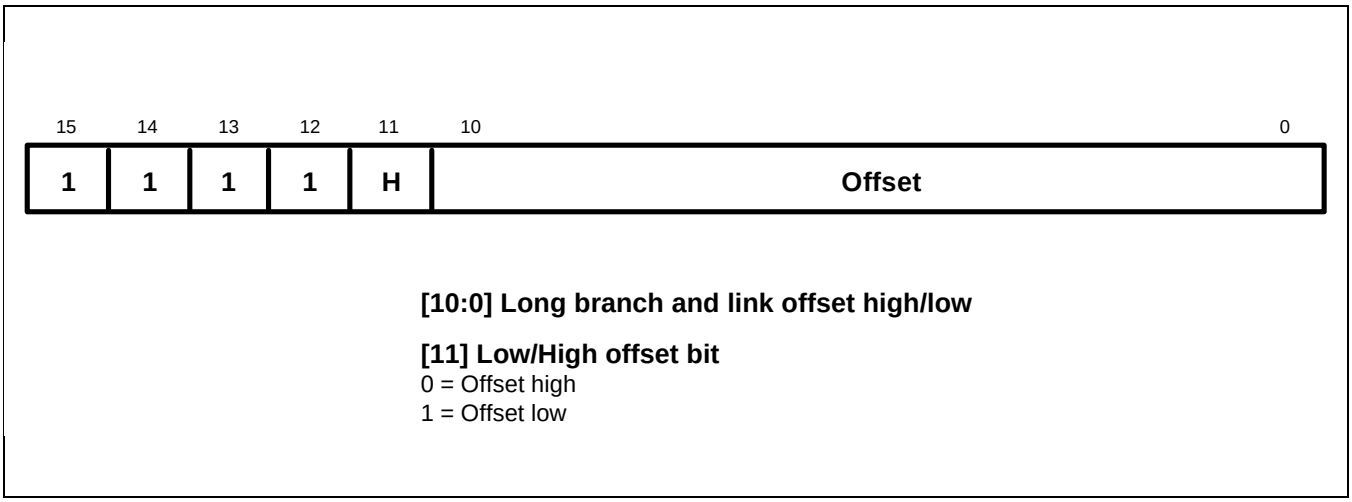


Figure 3-48. Format 19

OPERATION

This format specifies a long branch with link.

The assembler splits the 23-bit two's complement half-word offset specifed by the label into two 11-bit halves, ignoring bit 0 (which must be 0), and creates two THUMB instructions.

Instruction 1 (H = 0)

In the first instruction the Offset field contains the upper 11 bits of the target address. This is shifted left by 12 bits and added to the current PC address. The resulting address is placed in LR.

Instruction 2 (H =1)

In the second instruction the Offset field contains an 11-bit representation lower half of the target address. This is shifted left by 1 bit and added to LR. LR, which now contains the full 23-bit address, is placed in PC, the address of the instruction following the BL is placed in LR and bit 0 of LR is set.

The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction

INSTRUCTION CYCLE TIMES

This instruction format does not have an equivalent ARM instruction.

Table 3-26. The BL Instruction

H	THUMB assembler	ARM equivalent	Action
0	BL label	none	LR := PC + OffsetHigh << 12
1			temp := next instruction address PC := LR + OffsetLow << 1 LR := temp | 1

EXAMPLES

```
            BL faraway          ; Unconditionally Branch to 'faraway'
next        ...                 ; and place following instruction
                                ; address, ie 'next', in R14,the Link
                                ; register and set bit 0 of LR high.
                                ; Note that the THUMB opcodes will
                                ; contain the number of halfwords to offset.
faraway     ...                 ; Must be Half-word aligned.
```

INSTRUCTION SET EXAMPLES

The following examples show ways in which the THUMB instructions may be used to generate small and efficient code. Each example also shows the ARM equivalent so these may be compared.

MULTIPLICATION BY A CONSTANT USING SHIFTS AND ADDS

The following shows code to multiply by various constants using 1, 2 or 3 Thumb instructions alongside the ARM equivalents. For other constants it is generally better to use the built-in MUL instruction rather than using a sequence of 4 or more instructions.

Thumb		ARM

1. Multiplication by 2^n (1,2,4,8,...)

```
LSL     Ra, Rb, LSL #n          ;MOV Ra, Rb, LSL #n
```

2. Multiplication by 2^n+1 (3,5,9,17,...)

```
LSL     Rt, Rb, #n      ;   ADD Ra, Rb, Rb, LSL #n
ADD     Ra, Rt, Rb
```

3. Multiplication by 2^n-1 (3,7,15,...)

```
LSL     Rt, Rb, #n      ;   RSB Ra, Rb, Rb, LSL #n
SUB     Ra, Rt, Rb
```

4. Multiplication by -2^n (-2, -4, -8, ...)

```
LSL     Ra, Rb, #n      ;   MOV Ra, Rb, LSL #n
MVN     Ra, Ra          ;   RSB Ra, Ra, #0
```

5. Multiplication by -2^n-1 (-3, -7, -15, ...)

```
LSL     Rt, Rb, #n      ;   SUB Ra, Rb, Rb, LSL #n
SUB     Ra, Rb, Rt
```

Multiplication by any C = {2^n+1, 2^n-1, -2^n or -2^n-1} * 2^n
Effectively this is any of the multiplications in 2 to 5 followed by a final shift. This allows the following additional constants to be multiplied. 6, 10, 12, 14, 18, 20, 24, 28, 30, 34, 36, 40, 48, 56, 60, 62 .....

```
(2..5)                  ;   (2..5)
LSL     Ra, Ra, #n      ;   MOV Ra, Ra, LSL #n
```

GENERAL PURPOSE SIGNED DIVIDE

This example shows a general purpose signed divide and remainder routine in both Thumb and ARM code.

Thumb code

```
;signed_divide                          ;  Signed divide of R1 by R0: returns quotient in R0,
                                        ;  remainder in R1
;Get abs value of R0 into R3
            ASR     R2, R0, #31         ;  Get 0 or -1 in R2 depending on sign of R0
            EOR     R0, R2              ;  EOR with -1 (0xFFFFFFFF) if negative
            SUB     R3, R0, R2          ;  and ADD 1 (SUB -1) to get abs value

;SUB always sets flag so go & report division by 0 if necessary
            BEQ     divide_by_zero

;Get abs value of R1 by xoring with 0xFFFFFFFF and adding 1 if negative
            ASR     R0, R1, #31         ;  Get 0 or -1 in R3 depending on sign of R1
            EOR     R1, R0              ;  EOR with -1 (0xFFFFFFFF) if negative
            SUB     R1, R0              ;  and ADD 1 (SUB -1) to get abs value

;Save signs (0 or -1 in R0 & R2) for later use in determining ; sign of quotient & remainder.
            PUSH    {R0, R2}

;Justification, shift 1 bit at a time until divisor (R0 value) ; is just <= than dividend (R1 value). To do this shift
dividend ; right by 1 and stop as soon as shifted value becomes >.
            LSR     R0, R1, #1
            MOV     R2, R3
            B       %FT0
just_l      LSL     R2, #1
0           CMP     R2, R0
            BLS     just_l

            MOV     R0, #0              ;  Set accumulator to 0
            B       %FT0                ;  Branch into division loop

div_l       LSR     R2, #1
0           CMP     R1, R2              ;  Test subtract
            BCC     %FT0
            SUB     R1, R2              ;  If successful do a real subtract
0           ADC     R0, R0              ;  Shift result and add 1 if subtract succeeded

            CMP     R2, R3              ;  Terminate when R2 == R3 (ie we have just
            BNE     div_l               ;  tested subtracting the 'ones' value).
```

```
;Now fixup the signs of the quotient (R0) and remainder (R1)
                POP     {R2, R3}                ;   Get dividend/divisor signs back
                EOR     R3, R2                  ;   Result sign
                EOR     R0, R3                  ;   Negate if result sign = – 1
                SUB     R0, R3
                EOR     R1, R2                  ;   Negate remainder if dividend sign = – 1
                SUB     R1, R2
                MOV     pc, lr
```

ARM Code

```
signed_divide                                   ;   Effectively zero a4 as top bit will be shifted out later
                ANDS    a4, a1, #&80000000
                RSBMI   a1, a1, #0
                EORS    ip, a4, a2, ASR #32
;ip bit 31 = sign of result
;ip bit 30 = sign of a2
                RSBCS    a2, a2, #0

;Central part is identical code to udiv (without MOV a4, #0 which comes for free as part of signed entry
sequence)
                MOVS    a3, a1
                BEQ     divide_by_zero

just_l                                          ;   Justification stage shifts 1 bit at a time
                CMP     a3, a2, LSR #1
                MOVLS   a3, a3, LSL #1          ;   NB: LSL #1 is always OK if LS succeeds
                BLO     s_loop
div_l
                CMP     a2, a3
                ADC     a4, a4, a4
                SUBCS   a2, a2, a3
                TEQ     a3, a1
                MOVNE   a3, a3, LSR #1
                BNE     s_loop2
                MOV     a1, a4
                MOVS    ip, ip, ASL #1
                RSBCS   a1, a1, #0
                RSBMI   a2, a2, #0
                MOV     pc, lr
```

DIVISION BY A CONSTANT

Division by a constant can often be performed by a short fixed sequence of shifts, adds and subtracts.

Here is an example of a divide by 10 routine based on the algorithm in the ARM Cookbook in both Thumb and ARM code.

Thumb Code

```
udiv10                                  ;  Take argument in a1 returns quotient in a1,
                                        ;  remainder in a2
        MOV     a2, a1
        LSR     a3, a1, #2
        SUB     a1, a3
        LSR     a3, a1, #4
        ADD     a1, a3
        LSR     a3, a1, #8
        ADD     a1, a3
        LSR     a3, a1, #16
        ADD     a1, a3
        LSR     a1, #3
        ASL     a3, a1, #2
        ADD     a3, a1
        ASL     a3, #1
        SUB     a2, a3
        CMP     a2, #10
        BLT     %FT0
        ADD      a1, #1
        SUB     a2, #10
0
        MOV      pc, lr
```

ARM Code

```
udiv10                                  ;  Take argument in a1 returns quotient in a1,
                                        ;  remainder in a2
        SUB     a2, a1, #10
        SUB     a1, a1, a1, lsr #2
        ADD     a1, a1, a1, lsr #4
        ADD     a1, a1, a1, lsr #8
        ADD     a1, a1, a1, lsr #16
        MOV     a1, a1, lsr #3
        ADD     a3, a1, a1, asl #2
        SUBS    a2, a2, a3, asl #1
        ADDPL   a1, a1, #1
        ADDMI   a2, a2, #10
        MOV     pc, lr
```

NOTES

4 SYSTEM MANAGER

INTRODUCTION

The KS32C6400 System Manager has the following functions:

- Arbitrates bus access requests from several master blocks, based on a fixed priority.
- Provides the appropriate memory control signals for an external memory access. For example, if a master block such as DMA or CPU generates an address that corresponds to a DRAM bank, the System Manager's DRAM control block generates appropriate DRAM control signals such as nRAS, nCAS, address and data.
- Supports big-endian mode with efficiency for most graphic device drivers (see Figure 4-3).
- Compensates for differences in bus width for data flowing between the external data bus and the internal data bus.

SYSTEM MANAGER REGISTERS

The KS32C6400 microcontroller has the SFRs, Special Function Registers, to keep the system control information of system manager, cache, internal SRAM, DMA, UART blocks and so on. The SFRs have the SMR, System Manager Register files, to configure the external memory maps such as DRAM, SRAM, ROM and extra-I/O control.

By utilizing the SMR, you can specify the memory type, external bus width, access cycles, required control signal timings(nRAS, nCAS and so on), memory bank location and each memory bank size which has a very configurable address space. The SMR provides (or accepts) the control signals, addresses, and data that are required by external devices during normal system operation. There are eleven registers to control one ROM bank, two SRAM banks, two DRAM banks, four extra-I/O banks and DRAM Refresh.

The KS32C6400 provides up to 32M bytes of address space and each bank provides up to 2M half-word memory space because each bank has 21address pins and 16bit data width.

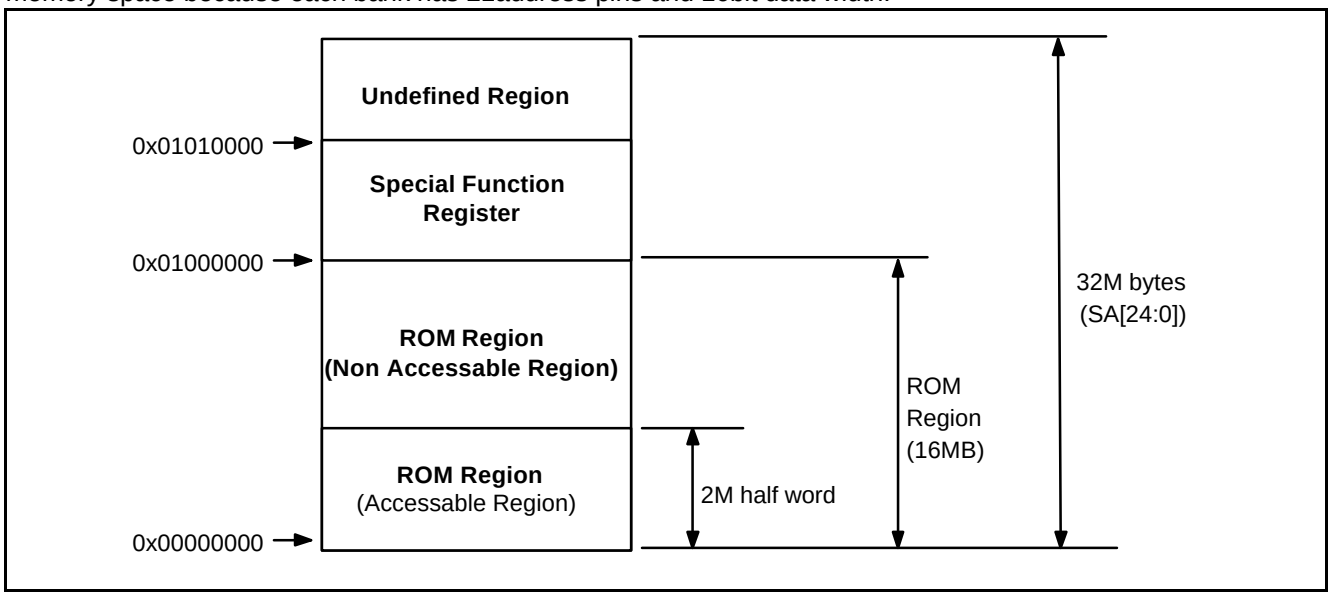


Figure 4-1. KS32C6400 Memory Map(Default Map after Reset)

The KS32C6400 provides 32MByte memory space and an internal 25-bit system address bus. You can use any address area from 0000000h to 1FEFFFFh by 64-Kbyte address steps. Each bank can be located anywhere in the 32MByte address space, except the upper 64-Kbyte area where SFRs (Special Function Registers) can be located. To use the full 32MByte memory space, we recommend for you to allocate the SFRs to the upper 64-kbyte address area, 1FF0000h-1FFFFFFh.

The configurable memory allocation in the KS32C6400 is designed to provide you with convenience. By manipulating the SMRs, you can easily allocate the memory area anywhere you wish and use the consecutively connected memory space without changing the H/W. You can also easily change the physical DRAM memory size by manipulating the SMR.

For example, if you want to change the size of memory space from 1M half word to 2M half word. You can enlarge the memory space just by changing the next pointer of the DRAM bank.

NOTE

The last 64 Kbyte area can not be allocated as memory banks except SFR. Because the last 64 Kbyte bank starts its address 1FFxxxxh, the next pointer of the last bank should be “200xxxxh”. Actually, the next point is 9 bits, so the value of the next pointer is 000xxxxh. If you need to utilize the full 32M bytes of memory space, it is recommended that you allocate the SFRs to the last 64-Kbyte area, 1F0000h-1FFFFFFh, and to use the remaining areas for other banks.

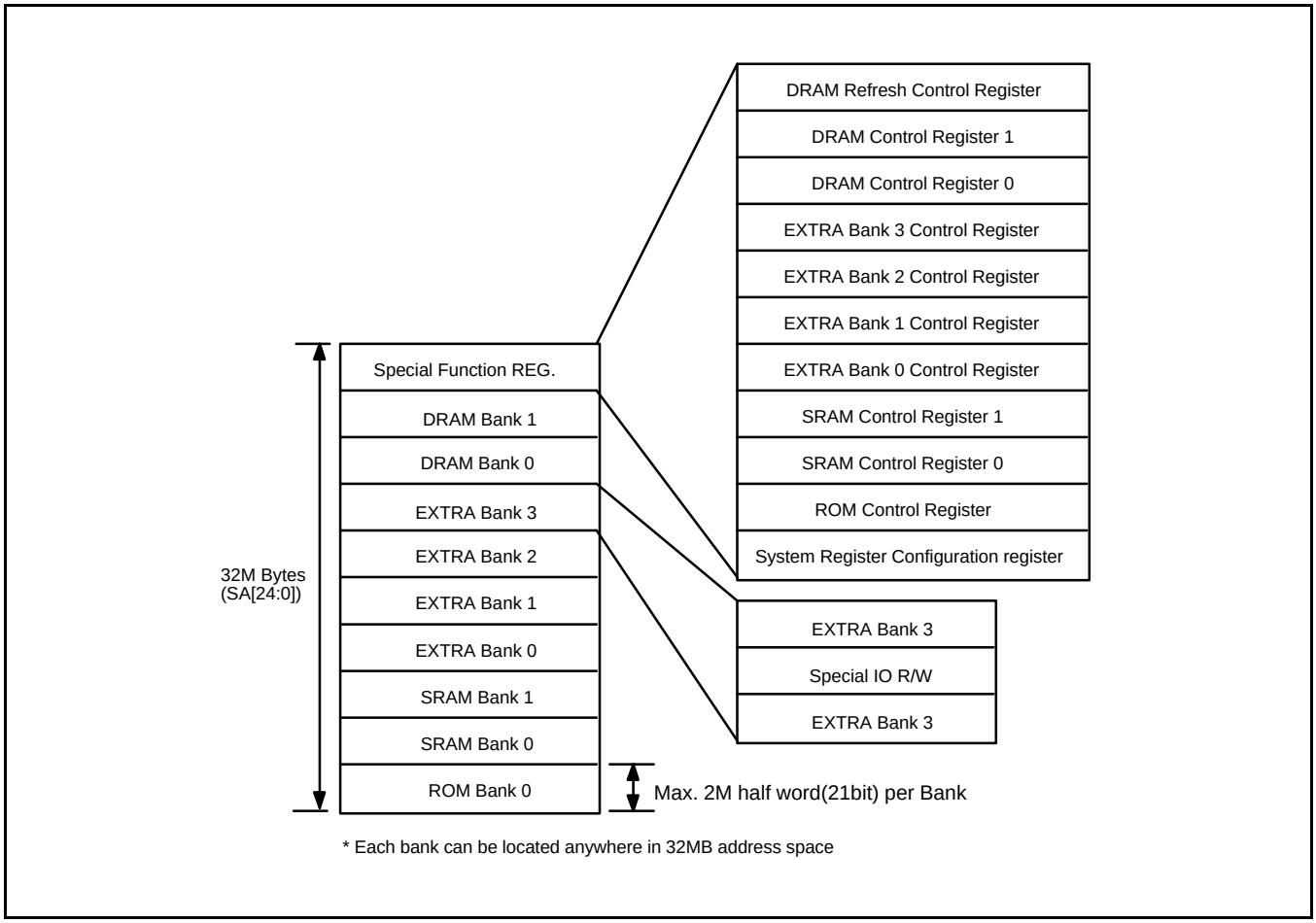


Figure 4-2. System Memory Map

SYSTEM REGISTER ADDRESS CONFIGURATION REGISTER (SYSCFG)

The SMRs (System Manager Registers) have the SYSCFG (System Register Address Configuration Register), which determines the start address (base point) of SFR (Special Function Register) files. The SYSCFG contains the start address of SFR. If the reset value of SYSCFG is 1001h, the SYSCFG is mapped to the virtual address 01000000h.

Register	Offset Address	R/W	Description	Reset Value
SYSCFG	0x0000	R/W	Special function register to determine the start address	0x1001

31 ~ 17	16	15	14	13	12 ~ 11 ~ 10 ~ 9 ~ 8 ~ 7 ~ 6 ~ 5 ~ 4	3	2	1	0
	CS	0	0	0	START ADDRESS	0	WE	CE	ST

[0] Stall Enable(ST)
0 = Disable. It's recommended for faster operation.
1 = Enable, Insert a internal wait inside the core logic when non-sequential memory accesses occur..

[1] Cache Enable(CE)
0 = Cache operation Disable
1 = Cache operation Enable

[2] Write buffer Enable(WE)
0 = Write buffer operation Disable
1 = Write buffer operation Enable

[12:04] SYSCFG Address (SFRs start address)
The resolution is 64KB, if you want to place the start addres at 1800000h, Setting value = 1800000h/10000h

[15:13] Reserved bits
User should fix '000'.

[16] Cahe Select bits
0 = 2K Cache Selection.
1 = 4K Cache Selection.

START ADDRESS

The SYSCFG[12:4] bits indicate the start address of SFRs. Since the SYSCFG is located at the bottom of the SFR file, the SYSCFG's location is the same as the start address of the SFRs.

You can allocate the SFRs to the arbitrary location by manipulating the SYSCFG. We recommend that you do not change the SYSCFG during operation once you have configured it after a system reset. The SYSCFG should not be overlapped with any other bank.

If the start address of the SYSCFG is changed, the other control registers in the SFRs will have the new address which is the sum of its offset address and the new address of SYSCFG. For example, after a system reset, the initial address of SYSCFG is 1000000h and the ROM control register has the initial address '1003000h, because the ROM control register has the offset value '3000h', and the initial address is the sum of 1000000h and 3000h. If the SYSCFG address is changed to 1800000h, the address of ROM control register will be 1803000h.

CACHE DISABLE/ENABLE

You can disable/enable the cache memory in the KS32C6400. You can enable the cache memory by setting the CE bit to logic 'one'. You can also use the non-cacheble area to maintain the data coherency of a specific memory area. Because the cache memory does not have the auto-flushing mode, you should be careful about the data coherency when you reenable the cache memory. You also have to check whether or not the DMA changes the memory data. The DMA accessible memory area must be non-cacheble to keep data coherency.

To keep the data coherency between the cache and the external memory, the KS32C6400 uses the write-through policy. To compensate for performance degradation, there is an internal four-depth write buffer. Please refer to Section 5 for more detail information.

WRITE BUFFER DISABLE/ENABLE

The KS32C6400 has four 'Write Buffer Registers' to enhance its memory writing performance. When the Write Buffer mode is enabled, the CPU writes data into the Write Buffer first, instead of an external memory which requires the longer memory write cycles. Write Buffer has 4 registers and each register includes a 32-bit data field, a 25-bit address field and a 2-bit status field.

STALL DISABLE/ENABLE

When the stall option is enabled, the MCU core logic inserts a wait, which occurs by non-sequential memory access. The MCU core has a larger time margin when memory access is executed. When the stall option is disabled, the MCU core logic does not insert a wait signal. The operation time of MCU core logic is faster than when the stall option is enabled.

CACHE SELECT 2K/4K

The KS32C6400 has 4KByte Cache and it is also used 2KByte Cache if you disable 4KByte Cache. If it is used 2KByte only, you can use the others 2KBytes Cache memory like internal SRAM. This bit must be setted before setting cache enable bit.

ROM CONTROL REGISTER

The KS32C6400 supports one ROM bank for program memory. The ROM bank has configurable features such as access timing, access size and page mode support. The ROMCON (ROM Control Register) in SMR supplies the control modes such as normal mode access, page mode access and wait cycles of each mode and external ROM bank.

The initial address of ROMCON is 01003000h which is the sum of the initial address of SYSCFG (01000000h) and ROM control register offset address (00003000h). The register address is reconfigurable. You can change the ROM control register by changing the content of SYSCFG. The real address of ROM control register is "SYSCFG address" + "Offset address" of ROM control register.

Register	Offset Address	R/W	Description	Reset Value
ROMCON	0x3000	R/W	ROM control register	0x80003002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NEXT POINTER									BASE POINTER									Tacc			Tacp		Pmc								DW	

[1:0] Bus Width (DW)
10 = 16 (half word) Others = No use
(The ROM interface supports only 16 bit of external data bus width)

[8:7] Page Mode Configuration (Pmc)
00 = Normal ROM 01 = 4 data page
10 = 8 data page 11 = 16 data page

[10:9] Page Mode Access Cycles (Tacp)
00 = 5 cycles 01 = 2 cycles
10 = 3 cycles 11 = 4 cycles

[13:11] Access cycles for ROM bank (Tacc)
000 = Disable bank 100 = 5 cycles
001 = 2 cycles 101 = 6 cycles
010 = 3 cycles 110 = 7 cycles
011 = 4 cycles 111 = Not used

[22:14] Start point of ROM bank (Base Pointer)
Indicates ROM bank start address

[31:23] End point + 1 of ROM (Next Pointer)
Indicates ROM bank end address + 1
(Next point value has to be bigger than Base point value, if Base point and Next point value are same, ROM bank is not valid anymore)

PAGE MODE ROM ACCESS (BURST MODE ACCESS)

The KS32C6400 ROM can interface with simple ROM and page mode ROM. You can enable/disable burst mode and can define the readable number of burst data by using ROMCON[8:7]. ROM has two different access cycles for simple ROM and page mode ROM. When page mode is selected, the first data access time is different from the access time of the following data if the data is in the same page.

Tacc, access cycles for ROM bank, is defined as the access cycle after the active bank changes to the ROM bank. This cycle time is also used for simple ROM access mode. When CPU reads consecutive data within the same page, the page mode ROM supplies a data read cycle shorter than reading the different page. The T acp bit in ROM control register defines the consecutive data read cycles in page mode ROM.

WRITING ON THE ROM BANK

In the KS32C6400, you can write data into the ROM bank area. Though the internal program of ROM is not changed physically, you may need a writing feature if SRAM or flash ROM is installed into the ROM bank.

ROM BANK SPACE

You can configure memory space in the KS32C6400. You can manipulate the memory bank size and bank location by using the ROMCON (ROM Control Register). The ROMCON has two 9-bit address pointers. One is the base pointer and the other is the next pointer. These two pointers denote the beginning and ending addresses of the ROM bank. The values of these pointers are compared with the address[24:16] to make a bank-select signal. The size of the ROM bank area can be increased/decreased by 64 K bytes. The value of the next pointer should be the sum of ROM bank end address and one.

For example, after a system reset, the start address of ROM bank is 00000000h and the end address is 00FFFFFFh. The value of the next pointer is 0100h ((0FFFFFFh+1h)/64-Kbytes). If the values of the next pointer and the base pointer of the ROM bank are the same, the ROM bank will be disabled.

INITIALIZATION

After a system reset, the initial value of ROMCON is set to 80003002h. In the system initialization, the external bus width is 16-bits (half-word) and the normal ROM mode is enabled. The longest value of the page mode access cycles is selected.

ROM PROGRAMMING

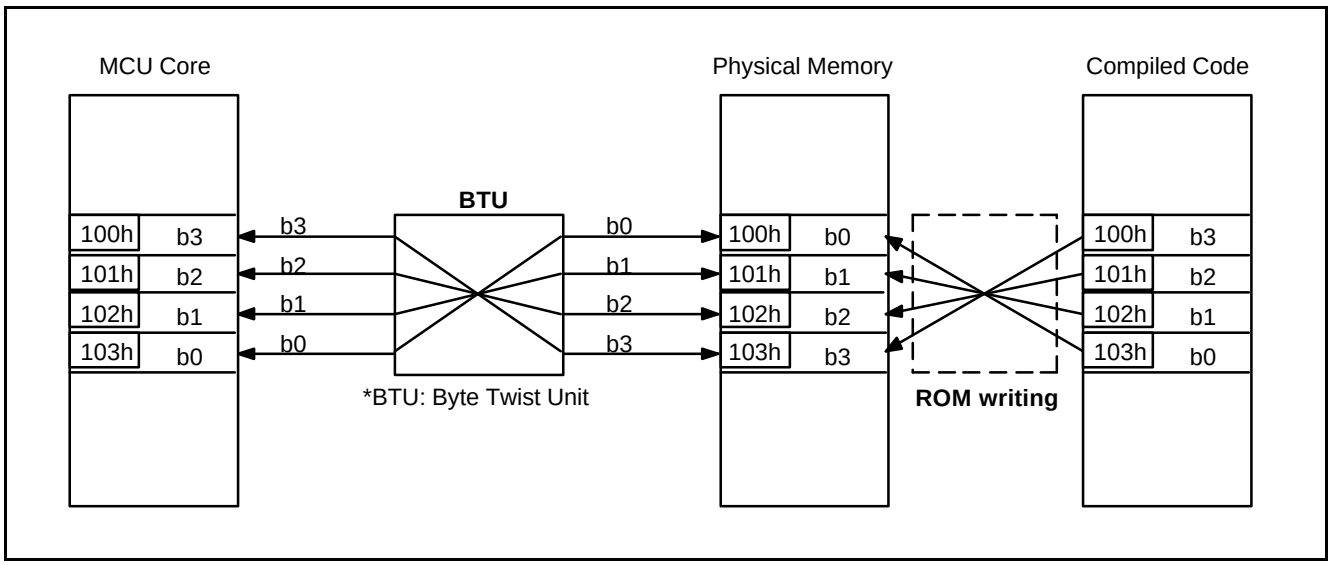


Figure 4-3. The Byte Swap Operation of BTU and Data Positions in Memory

BIG-ENDIAN FORMAT / LITTLE-ENDIAN FORMAT

In the Big-Endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. The byte '0' of the memory system is connected to data lines from 31 to 24.

In the Little-Endian format, the lowest numbered byte in a word is considered as the least significant byte, and the highest numbered byte as the most significant. The byte '0' of the memory system is connected to data lines from 7 to 0.

BIG-ENIDAN SUPPORTING CORE AND LITTLE-ENGINE SUPPORTING PHYSICAL MEMORY

The ARM7TDMI core in KS32C6400 is a little-endian base core supporting the big-endian format. To support the big-endian format in ARM7TDMI, the KS32C6400 adopts BTU, Bus Twist Unit. The main function of BTU is to exchange bytes in a word as shown in Figure 4-5. For example, when MCU core accesses the data ‘100h’, the BTU alters ‘100h’ to ‘103h’, and the CPU will get the data, 103h, in the physical memory.

To put big-endian format data into memory, a compiled code has to put into memory ROM by exchanging bytes in a word. In Figure 4-5, double swappings (BTU and compiled code swapping) enable Big-Endian format with no problem. The reason that the KS32C6400 uses the double swappings is due to internal hardware implementation issue.

BYTE SWAPPING IN A WORD

Byte swapping is executed by using the sample C. The sample changes the byte sequence in a word.

Unsigned Long Swap (Unsigned long Data) // Make the Sequence of Bytes Reverse in a word

```
{
        return      ( ( (0xff000000 & data)>>24)+
                      ( (0x00ff0000 & data)>>8)+
                      ( (0x0000ff00 & data)<<8)+
                      ( (0x000000ff & data)<<24) ) ;
}
```

ROM WRITING

BTU changes the byte sequence in a word. Program codes are byte-swapped. To write program to ROM, do as follows;

1. Compile the program in big-endian format
2. Byte-swap the compiled code (recommend to make a program that executes byte-swap of a binary file)
3. Write the code to ROM

LITTLE-ENDIAN FORMAT CODE VERSUS BYTE-SWAPPED BIG-ENDIAN FORMAT CODE

If character strings do not exist in a program, the Little-Endian format code may be the same as the byte-swapped big-endian format code. Because the bytes in strings are not affected by the endian format, the two codes are different in strings. The byte-swapped big-endian format code has to be used in the KS32C6400. If Little-Endian format code is used, the strings will not be displayed correctly (byte-swapped strings may be displayed).

INTERFACING EXTERNAL PERIPHERALS

Peripheral address is also byte-swapped. If you want to access address 00b in memory, you have to access address 11b in the program. This is due to the word swapping of BTU. The relation between physical address and the address used by instructions is as follows.

Table 4-1. The Relation Between Physical Address and Address in Instructions

Physical address	Byte Wide Access (Address Used in Instructions)	Half Word Wide Access (Address Used in Instructions)
00b	11b	10b
01b	10b	–
10b	01b	00b
11b	00b	–

SRAM CONTROL REGISTERS

The KS32C6400 has two banks of SRAM. Each bank can set up its SRAM access configuration. SRAMCON0 and SRAMCON1 (SRAM Control Registers) specify not only the features of SRAM but also one special I/O in the external bank 3.

The initial address of SRAMCON0,1 are 01003004h and 01003008h. The real address of each SRAM control registers is the sum of 'SYSCFG address' and 'offset address' of each SRAM control register. The register address is reconfigurable, so you can change the SRAM control register address using the SYSCFG.

Register	Offset Address	R/W	Description	Reset Value
SRAMCON0	0x3004	R/W	SRAM control register 0	0x00000000
SRAMCON1	0x3008	R/W	SRAM control register 1	0x00000000

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0			
NEXT POINTER	BASE POINTER	Tacc	SPECIAL I/O ADDRESS	DW

[1:0] Bus Width (DW)
00 = Disable bank 01 = 8(byte)
10 = 16 (half word) 11 = No used
(KS32C6400 has external 16bit data bus and supports 8/16bit data bus width.)

[10:2] Special I/O address setting value
It denotes the start address of Special I/O address in extra bank 3. Extra bank 3 has one Special I/O areas for cost effective solution.
(see Extra I/O control register bank explanation for more information)

[13:11] Access cycles for SRAM bank (Tacc)
000 = Disable bank 100 = 5 cycles
001 = 2 cycles 101 = 6 cycles
010 = 3 cycles 110 = 7 cycles
011 = 4 cycles 111 = Not used

[22:14] Start point of SRAM bank (Base Pointer)
Indicates SRAM bank start address

[31:23] End point + 1 of SRAM (Next Pointer)
Indicates SRAM bank end address + 1
(Next point value has to be bigger than Base point value, if Base point and Next point value are same, ROM bank is not valid anymore)

SRAM BANK SPACE

The KS32C6400 provides two SRAM banks and each bank can be configured differently. You can program SRAM access cycles, memory bank size and location by using the SRAMCON0,1. The SRAM control register has two 9-bit address pointers (base pointer and next pointer). These two pointers denote the start address and end address of the SRAM bank. These 9 bits are compared with the address [24:16] to make the bank-select signal. The size of SRAM bank area can be increased/decreased by 64 K bytes. The value of the next pointer should be the sum of the end address of SRAM bank and one.

INITIALIZATION

When the system is initialized, the value of both SRAM control registers is 0000000h and it specifies that SRAM banks are disabled because the next pointer and the base pointer have the same values.

SPECIAL I/O ADDRESS

The extra-bank 3 of the KS32C6400 has one special I/O area to make simple control signal for the external latch. SRAM control register 0 has dedicated 9 bits for this special I/O areas in the extra-bank 3.

Extra-bank 3 provides two special control signals, nIORD and nIOWR. When you read/write data from/to external latch devices, the two special control signals do not need additional address decoding logic. These signals are available only at extra-bank 3. When MCU accesses any special I/O area(64KB, 16bit offset address) specified by SRAM control registers, the extra-bank interface logic generates a I/O read/write signals to the corresponding area. Figure 4-8 shows the diagram of special I/O read/write interface logic.

ADDRESS BUS GENERATION

The address bus of KS32C6400 is quite different from the general MCUs'. Although general MCU does not use the A0 pin in 16-bit data bus width, the KS32C6400 always uses the A0 pin regardless of data bus width. When an 8-bit data bus is selected, the resolution of address bus is a byte and when a 16-bit is selected, the resolution of address bus is a half-word.

Data Bus Width	External Address Pins (ADDR[20:0])	Accessible Memory size
8-bits	A20-A0 (internal)	4M bytes
16-bits	A21-A1 (internal)	4M half-word

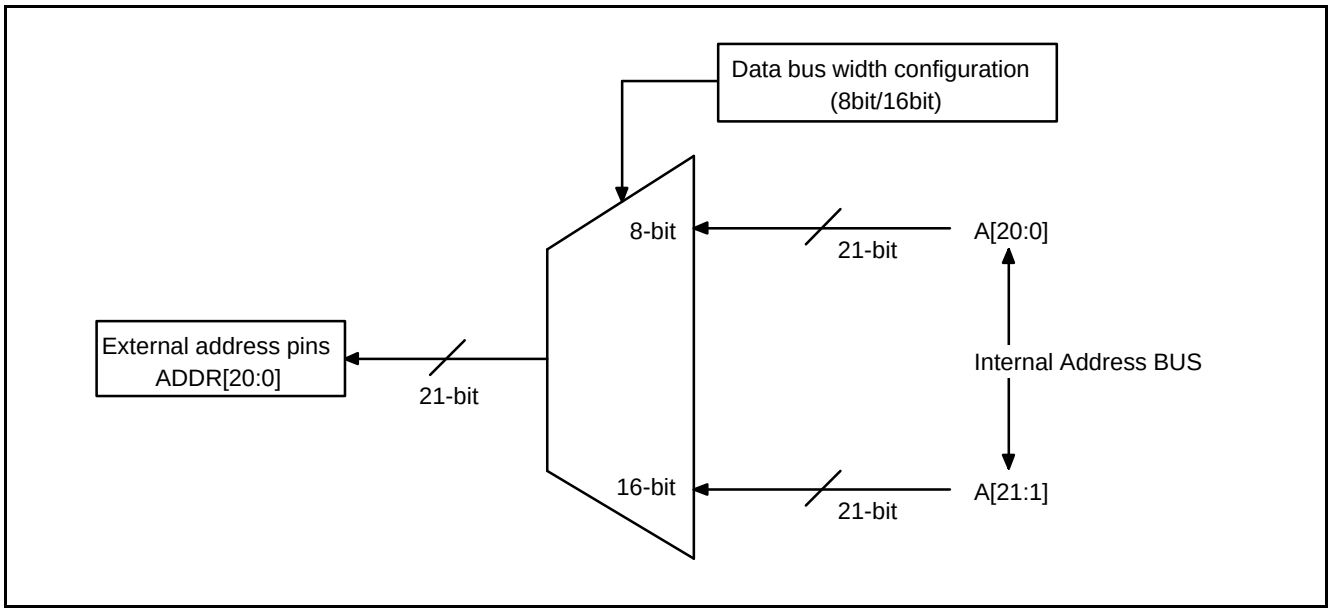


Figure 4-4. External Address Bus Generation (ADDR[20:0])

DRAM CONTROL REGISTERS

The KS32C6400 has two banks of DRAM and each bank can control DRAM access timing as other memory banks do. The DRAM interface has two DRAMCON0,1 (DRAM Control Registers) and one REFCON (DRAM Refresh Control Register). The initial addresses of each DRAM control register are 0100301Ch and 01003020h. The Refresh control register address is 01003024h. You can change the address of the DRAM control register by changing the value of SYSCFG.

Register	Offset Address	R/W	Description	Reset Value
DRAMCON0	0x0000301C	R/W	DRAM 0 control register	0x00000000
DRAMCON1	0x00003020	R/W	DRAM 1 control register	0x00000000

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12	11	10 9 8	7	6 5	4	3 2	1 0
NEXT POINTER	BASE POINTER	Trp	Trc	Tcs	Tcp	Tpgm	EDO	CAN	DW

[1:0] DW — Bus Width
00 = Disable bank 01 = 8(byte)
10 = 16 (half word) 11 = No used

[3:2] CAN — Column Address Number
00 = 8bits 01 = 9bits
10 = 10bits 11 = 11bits

[4] EDO — EDO DRAM or Ordinary DRAM
0 = Ordinary 1 = EDO DRAM

[6:5] Tpgm — CAS strobe time (@ Page mode)
00 = 1cycle 01 = 2cycles
10 = 3cycles 11 = 4cycles

[7] Tcp — CAS pre-charge
0 = 1cycle 1 = 2cycles

[10:8] Tcs — CAS strobe time (@ Single mode)
000 = 1 cycle 100 = 5 cycles
001 = 2 cycles 101 = Not used
010 = 3 cycles 110 = Not used
011 = 4 cycles 111 = Not used

[11] Trc — RAS to CAS delay
0 = 1 cycle 1 = 2 cycles

[13:12] Trp — RAS pre-charge time
00 = 1 cycle 01 = 2 cycles
10 = 3 cycles 11 = 4 cycles

[22:14] Base Pointer — DRAM bank start address.

[31:23] Next Pointer — DRAM bank end address + 1.

The KS32C6400 provides a fully programmable configuration for DRAM interface. You can easily modify the configuration setting value such as external data bus width, number of access cycles for fast page or EDO, access cycles for each DRAM bank and row address strobe (nRAS) pre-charge timing by changing the value of the corresponding DRAM control register. The Refresh Control register controls the DRAM refresh operation. The KS32C6400 supports CAS before the RAS (CBR) refresh mode and Self-Refresh mode.

The KS32C6400 can generate row and column address, and supports symmetric/asymmetric address DRAM by changing the number of address line from 8 to 11. It can support various sizes of DRAM by varying column address size. If the number of a column address or a row address is larger than 11, the accessible DRAM memory size is smaller than the original size of the DRAM. For example, if a 16M-bit DRAM with 4M x 4 (row address = 12-bit and column address=10-bit) is connected to KS32C6400, the maximum accessible size of the memory is 8M-bit (11-bit x 10-bit) and the other 8M-bit will be obsolete.

EDO MODE DRAM ACCESS

Even if you specify DRAM as EDO mode, the KS32C6400 gives the same timing diagram compared with normal fast page mode. However, the KS32C6400 CPU fetches data (when read) a half-clock later than normal fast page mode, because EDO mode can make data valid even if CAS goes to high when RAS is low. CPU can have enough time to access and latch the data. Eventually, EDO mode can reduce memory access time.

DRAM BANK SPACE

The KS32C6400 provides two DRAM banks and each bank can be configured differently. You can program the DRAM access cycles, memory bank size and bank location by using two DRAM control registers, DRAMCON0,1. DRAM control register has two 9-bit address pointers, Base and Next pointer. These two pointers denote the start and end address of DRAM bank. These 9-bits are compared with the address [24:16] to make the bank select signal. The size of the DRAM bank area can be increased/decreased by 64 K bytes. The value of the next pointer should be the sum of the end address of the bank and one.

INITIALIZATION

When the system is initialized, the start address and end address of two DRAM bank are 00000000h. It specifies that the DRAM bank is disabled because the respective values of the next pointer and the base pointer are the same.

DRAM BANK CONFIGURATION

The DRAM has different write methods from SRAM or other external memories. DRAM module has two CAS signals to separate data bus by byte order. RAS signal is used for bank selection and CAS signal is used for byte selection.

Ex) Setting for 60 ns EDO DRAM (KM416V1204)

Conditions	Setting Value for DRAMCON
Memory map: 1000000h - 11FFFFFh DRAM: 10 bits (row)x 10 bits (column) x 16 bits (data), 60 ns, EDO MCLK: 33 MHz	0x9040101a

DRAM REFRESH CONTROL REGISTER

The KS32C6400 provides the CAS Before RAS (CBR) refresh and Self-refresh Mode. The refresh control register (REFCON) determines refresh mode, refresh timings, refresh intervals as well as external bus enable.

Register	Offset Address	R/W	Description	Reset Value
REFCON	0x00003024	R/W	DRAM refresh control	0x00000001

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REFRESH COUNT											Tscr	Tcs			REN	0																VSF

[0] VSF
Validity of Special Register Field
0 = Not Accessable to Memory Bank
1 = Accessable to Memory Bank
(Whenever MCU access one of System manager registers(SMR), VSF bit is auto-cleared and external bus is disabled. To re-activate external bus, VSF bit should be set to 1 by using STMIA instruction in the last. Programmer should write into 10 system manager registers altogether with STMIA instruction while VSF bit set instruction is in the last.)

[15] 0
Reserved '0'

[16] REN
Refresh Enable
0 = Self refresh mode or Disable DRAM refresh. (When this bit is set '0', DRAM enters the self refresh mode and cannot be accessed.)
1 = Enable DRAM Refresh. (When this bit is cleared, MCU refreshs DRAM periodically and can read/write the DRAM)

[19:17] Tcs
CAS Hold Time
000 = 1 cycle 100 = 5 cycles
001 = 2 cycles 101 = Not used
010 = 3 cycles 110 = Not used
011 = 4 cycles 111 = Not used

[20] Tcsr
CAS Set-up Time
0 = 1 cycle 1 = 2 cycles

[31:21] Refresh count
Refresh interval
Refresh Period = $(2^{11}$ - Value + 1)/MCLK
Example) if refresh duration is 15.6us,

Refresh count Value = $(2^{11}$ + 1) - (33MHz x 15.6us)
= 10111111110b

DRAM SELF-REFRESH MODE

DRAM requires a refresh operation periodically to keep data correct, and JEDEC defines a couple of refresh modes. The self-refresh mode, which is defined in JEDEC specification, enables the DRAM to refresh memory cells internally without periodical external refresh control signals, unless another refresh mode happens or power fails.

The operation of self-refresh is similar to that of CBR (CAS before RAS) refresh. Once after CPU generates CBR mode signals and keeps the CBR mode state for more than 100 us, DRAMs the recognize refresh mode as self-refresh mode instead of CBR refresh.

SELF-REFRESH MODE BY HARDWARE

When a nRESET, system reset pin, is low, the system manager block generates self-refresh mode signals. For example, whenever the KS32C6400 is initialized, it activates self-refresh mode. This hardware refresh feature enables the system to avoid DRAM data loss if a system back-up circuitry supplies power to DRAM continuously while main power is disconnected.

When the main power of the system is disconnected, the KS32C6400 will be disabled. Meanwhile, if DRAM has power back-up circuitry, it still requires periodical refresh signals from the KS32C6400. Therefore, if KS32C6400 does not make DRAM self-refresh mode, it loses valid DRAM data in a short time.

For this reason, when the main power of the system is disconnected and nRESET goes to the low signal, the system manager block of the KS32C6400 makes self-refresh signals. You can make memory back-up systems easily by utilizing this feature, if only DRAM is used for system memory.

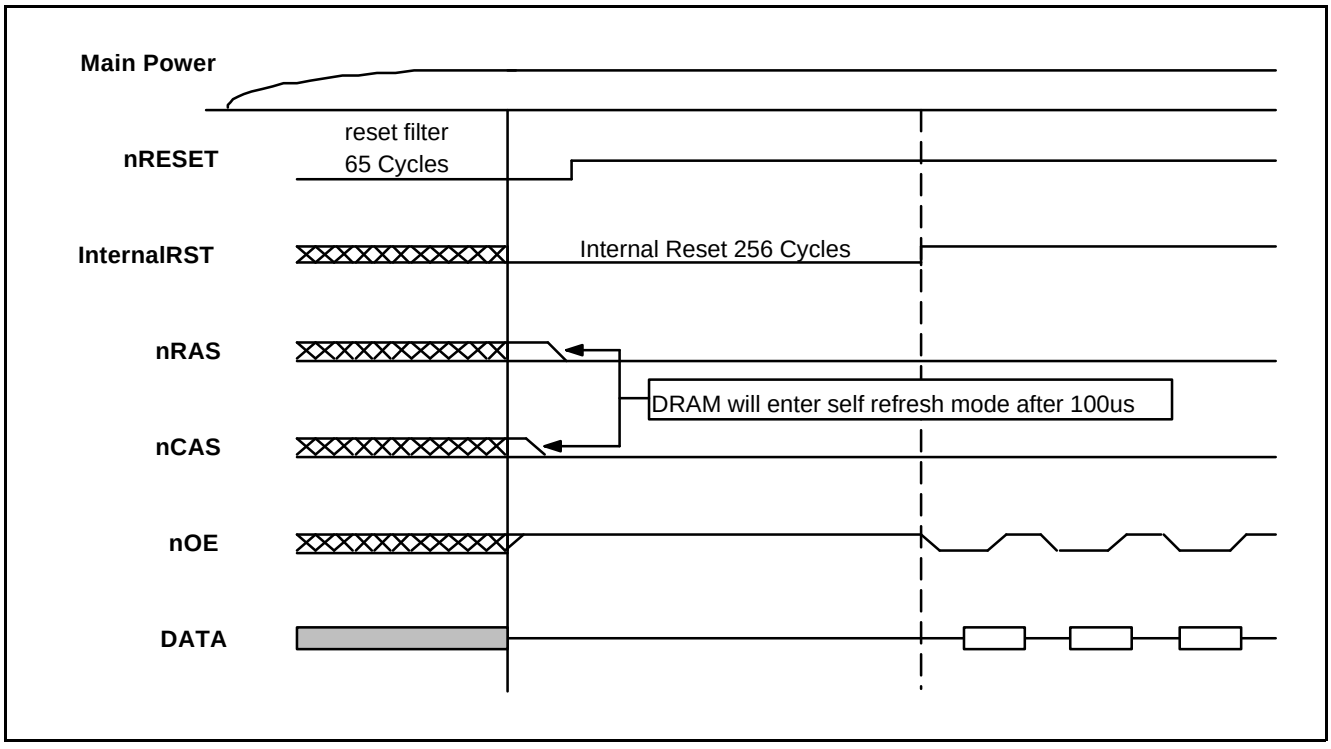


Figure 4-5. Self Refresh Mode Entry Process by nRESET

SELF-REFRESH MODE BY SOFTWARE

After a system reset, the KS32C6400 goes into the DRAM self-refresh mode. By setting the REN bit of DRAM refresh control register to "1", you can make the system manager block work as a normal DRAM access mode.

To enable the self-refresh mode during a normal DRAM access mode, you need to change the REN bit to "0". System manager detects the value of the REN bit, changing the value from 1 to 0, and then activates the self-refresh mode. If you want change the mode from self-refresh mode to normal DRAM access mode, just write "1" to REN bit once again.

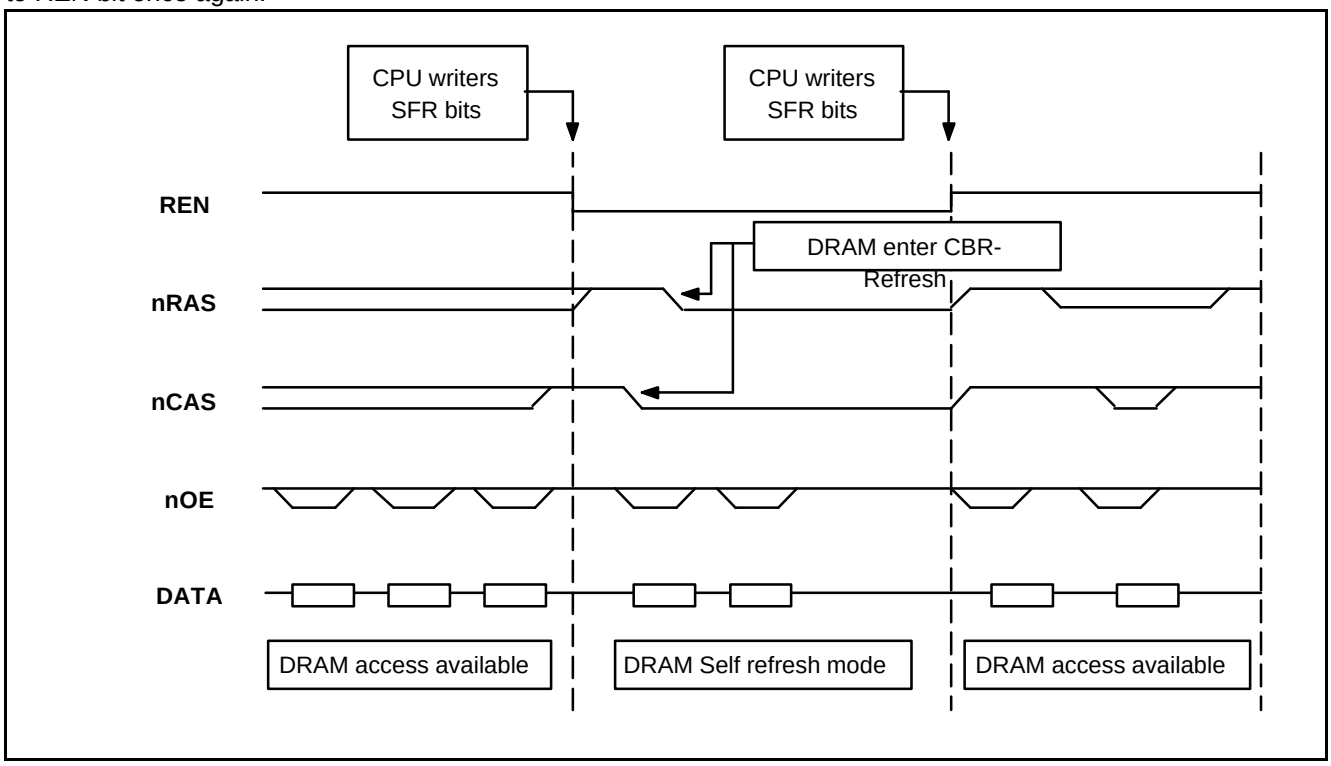


Figure 4-6. Self Refresh Mode Entry Process by Software

NOTE:

1. When DRAM does not recognize self-refresh mode after a system initialization

Even though the KS32C6400 activates self-refresh mode when the system power is connected, DRAM may not recognize self- refresh mode correctly, because of unstable state of control signals during a system initialization (most of DRAMs recognize the self-refresh mode very well at power on). when it happens, the KS32C6400 may fectch corrupted data from external memories because DRAM and ROM share OE(Output Enable) signal and they may generate data altogether.

The KS32C6400 has a watch-dog timer to cope with a system malfunction problem. When the KS32C6400 is initialized, the watch-dog timer is enabled and makes the external system reset signal unless MCU disables the watch-dog timer in the middle of an operation. Therefore, it is recommended that you put the code, which disables the watch-dog timer, into the boot ROM area. If "power on initial" does not work correctly and KS32C6400 fetches corrupted data, the watch-dog timer will make a system reset signal, causing the system reset of the KS32C6400 once again. The second watch-dog reset will cause DRAM to enter the self-refresh mode since system power and other states are stable.

2. DRAM access during self refresh mode

If the KS32C6400 accesses external DRAM to read or write data during the self-refresh mode, the nRAS and nCAS signals do not working at all, because the DRAM accessing during the self-refresh mode causes corrupted data to be read or written.

MEMORY ACCESS IS FORBIDDEN WHEN THE SMR IS CHANGED.
The external bus is disabled when MCU accesses SMRs to change system memory configurations. It is intended to prevent the system malfunction, caused by memory address space overlapped during the new configuration. To re-activate external bus operation, the VSF bit in the refresh control register needs to be set to logic 'one' by writing SMRs with STMIA instruction. While STMIA instruction writes 10 registers of SMRs, the refresh control register must be written at the last step with VSF bit = “0” so that the external bus can be reactivated right after the System Manager Register has a new configuration.

It is not recommended for you to change the SFRs after a system initialization. If the SFRs is changed, especially memory related areas, you have to flush the cache memory for data coherency.

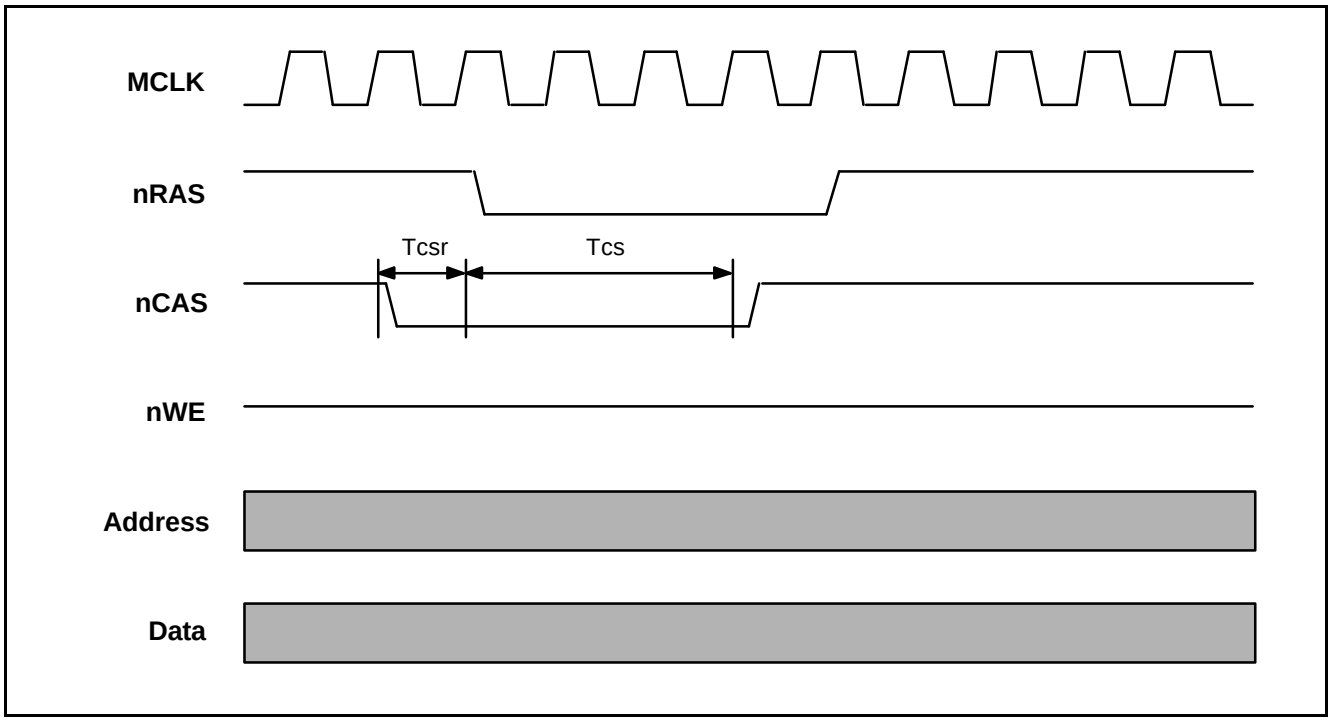


Figure 4-7. DRAM Refresh Timing

EXTRA-BANK ACCESS CONTROL REGISTERS

The KS32C6400 provides four extra-banks and four EXTCONn (extra-bank Control registers) which control timing, bank size and bus width. Extra-bank 3 has special features compared with other extra-banks. It has one special dedicated addresses (refer to SRAM control register) to provide the low cost external I/O control solution. Extra-bank 3 has special signals such as nIORD and nIOWR. When you read/write data from/to external latch devices in extra-bank 3, the extra-address decoding ICs are not required any more. Basically, they have the same timing diagram as extra-bank 3.

The initial address of each I/O control register is the sum of its own offset address with the initial SYSCFG register address, 01000000h.

Register	Offset Address	R/W	Description	Reset Value
EXTCON0	0x300C	R/W	Extra-bank 0 control register	0x00000000
EXTCON1	0x3010	R/W	Extra-bank 1 control register	0x00000000
EXTCON2	0x3014	R/W	Extra-bank 2 control register	0x00000000
EXTCON3	0x3018	R/W	Extra-bank 3 control register	0x00000000

31 30 29 28 27 26 25 24 23	22 21 20 19 18 17 16 15 14	13 12 11	10 9 8	7 6 5	4 3 2	1 0
NEXT POINTER	BASE POINTER	Tacc	Tcoh	Tacs	Tocs	DW

[1:0] DW — Programmable Bus Width
00 = Disable bank 10 = Half Word
01 = Byte 11 = No used

[4:2] Tcos — Chip Selection Set-up on nOE
000 = 0 cycle 100 = 4 cycles
001 = 1 cycles 101 = 5 cycles
010 = 2 cycles 110 = 6 cycles
011 = 3 cycles 111 = 7 cycles

[7:5] Tacs — Address Set-up before nECS
000 = 0 cycle 100 = 4 cycles
001 = 1 cycles 101 = 5 cycles
010 = 2 cycles 110 = 6 cycles
011 = 3 cycles 111 = 7 cycles

[10:8] Tcoh — Chip Selection Hold on nOE
000 = 0 cycles 100 = 4 cycles
001 = 1 cycles 101 = 5 cycles
010 = 2 cycles 110 = 6 cycles
011 = 3 cycles 111 = 7 cycles

[13:11] Tacc — Access cycles (nOE low time)
000 = Not Used 100 = 5 cycles
001 = 2 cycles 101 = 6 cycles
010 = 3 cycles 110 = 7 cycles
011 = 4 cycles 111 = 8 cycles

[22:14] Base Pointer — Start Address of Extra Bank n.
It denotes the start address of Extral bank x by word unit.

[31:23] Next Pointer — End point + 1 of Extra Bank n.
It denotes the start address of next Extra bank.

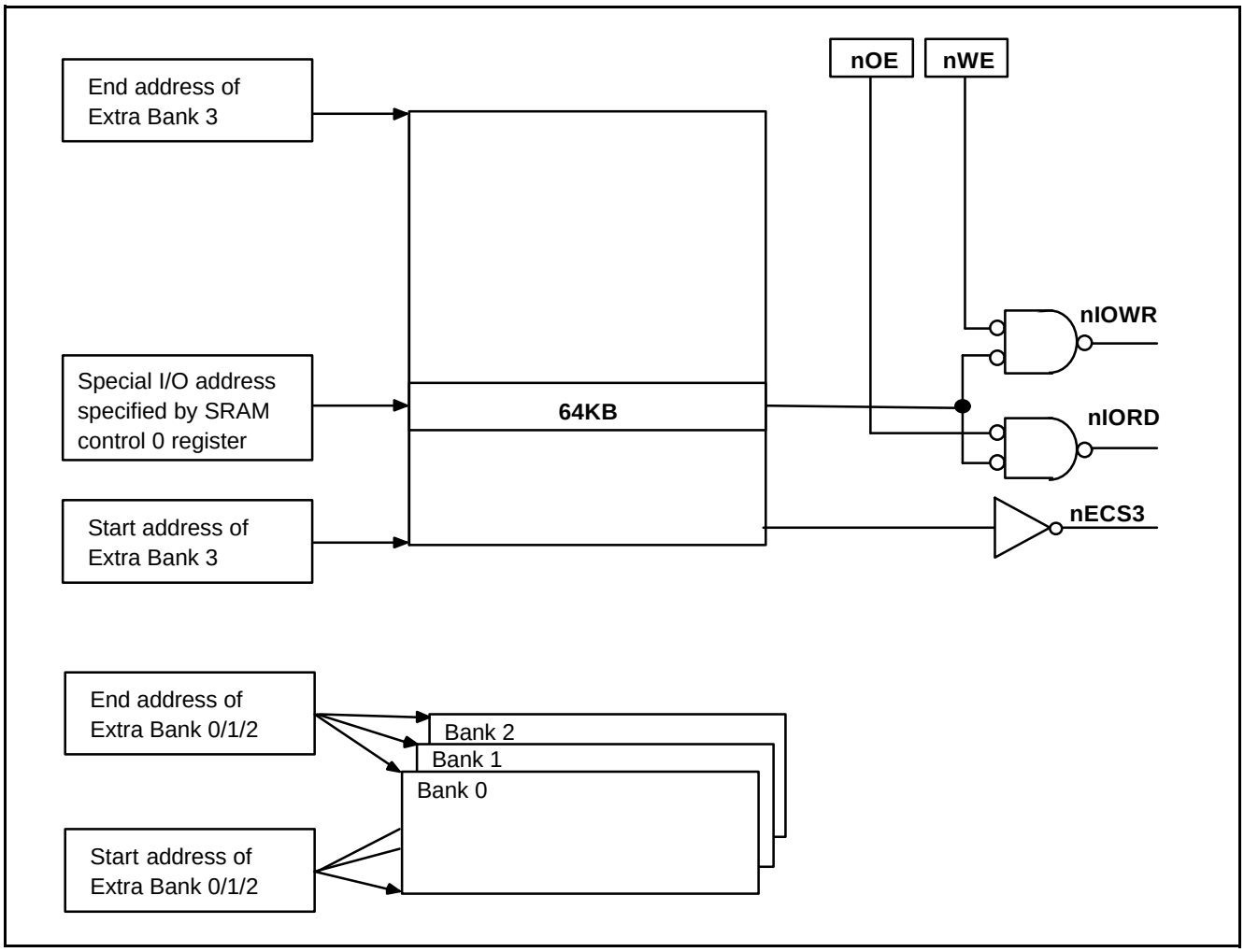


Figure 4-8. Special I/O Address Map

FETCHING DATA FROM READ CYCLE

When fetching data, the point of data reading is the last down edge of MCLK within the nECS active region. Users may be curious about the figure 4-16, nOE's deasserting before the point of data reading. If nOE has to be deasserted after the point of data reading, use 'tcoh'=0 which defines the time between nOE's deasseting and nECS's deasserting. Setting 'tcoh' as 0, nOE is deasserted after the point of data reading as you want.

SPECIAL I/O ADDRESS

SRAM control register 0 has dedicated 9 bits for the extra-bank 3, for providing the low cost system solution. Bank 3 has special signals, nIORD, nIOWR. When you read/write data from/to external latch devices, these signals prevent extra-address decoding ICs. These signals are only available at extra-bank 3. When CPU access any of the special I/O address area (64-Kbytes, 16-bit offset address) specified by SRAM Control register 0, the extra-bank generates I/O Read and Write signals for the corresponding address area. Fig 4-16, 17 shows the timing diagram of special I/O read/write cycles.

MEMORY MAPPING FOR EXTERNAL MEMORY I/O

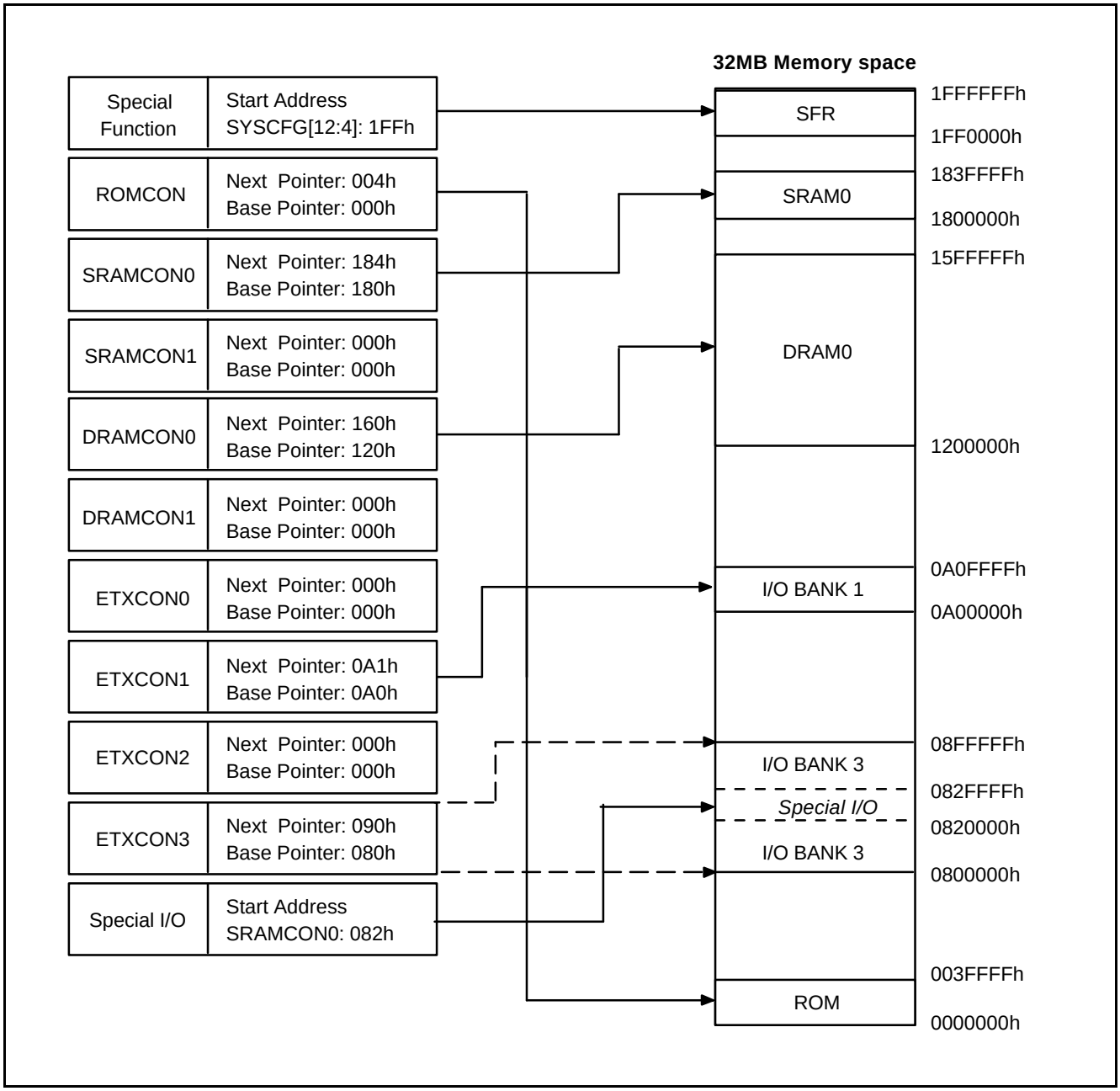


Figure 4-9. An Example of System Manager Register Setting

TIMING DIAGRAM

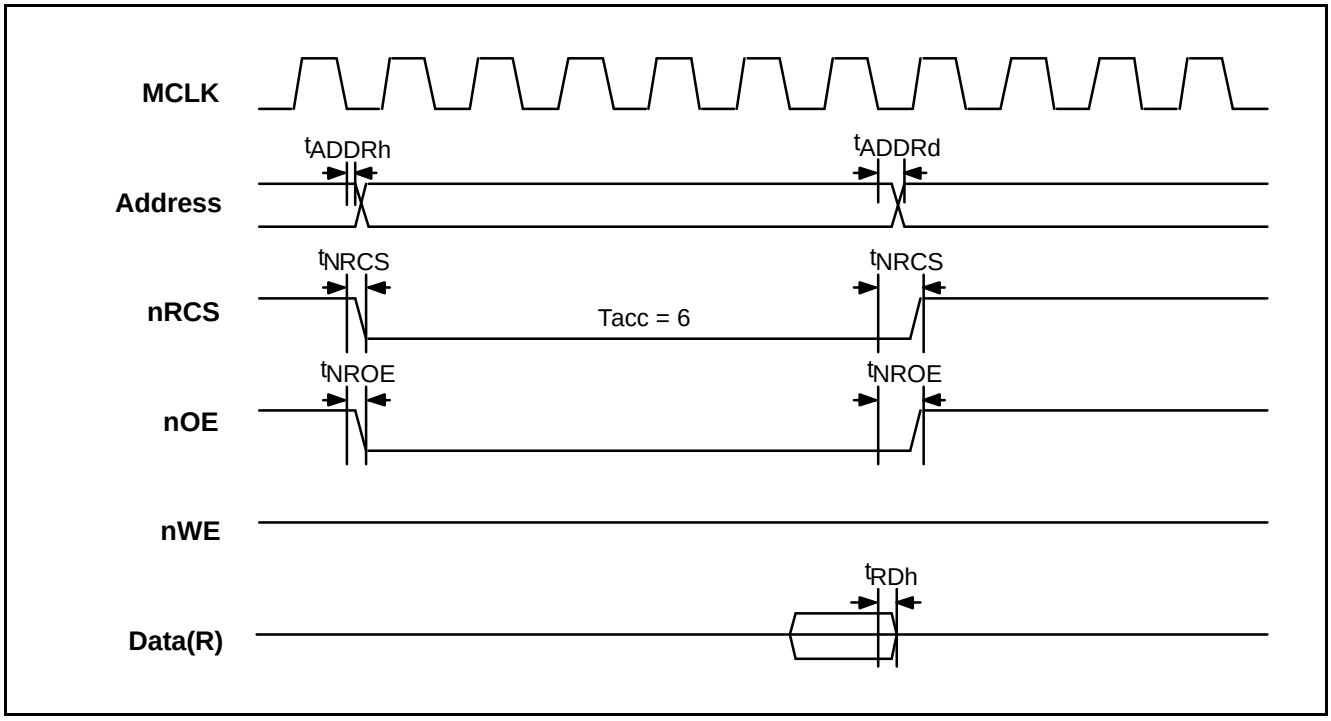


Figure 4-10. Simple ROM Access Timing

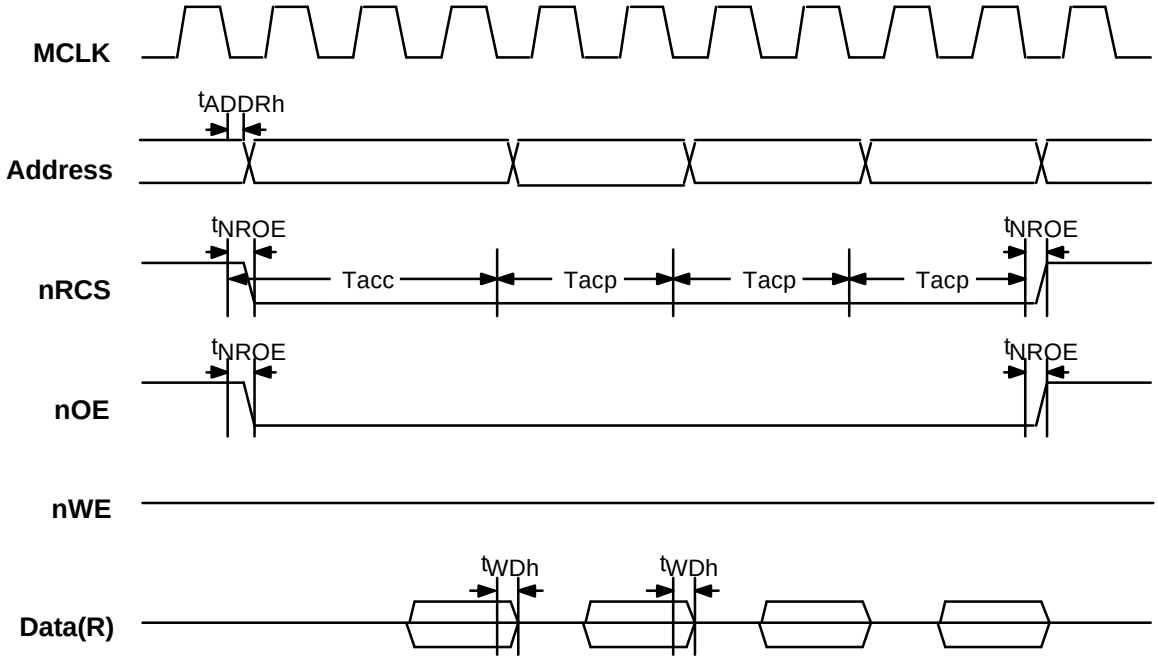


Figure 4-11. Page Mode ROM Access Timing

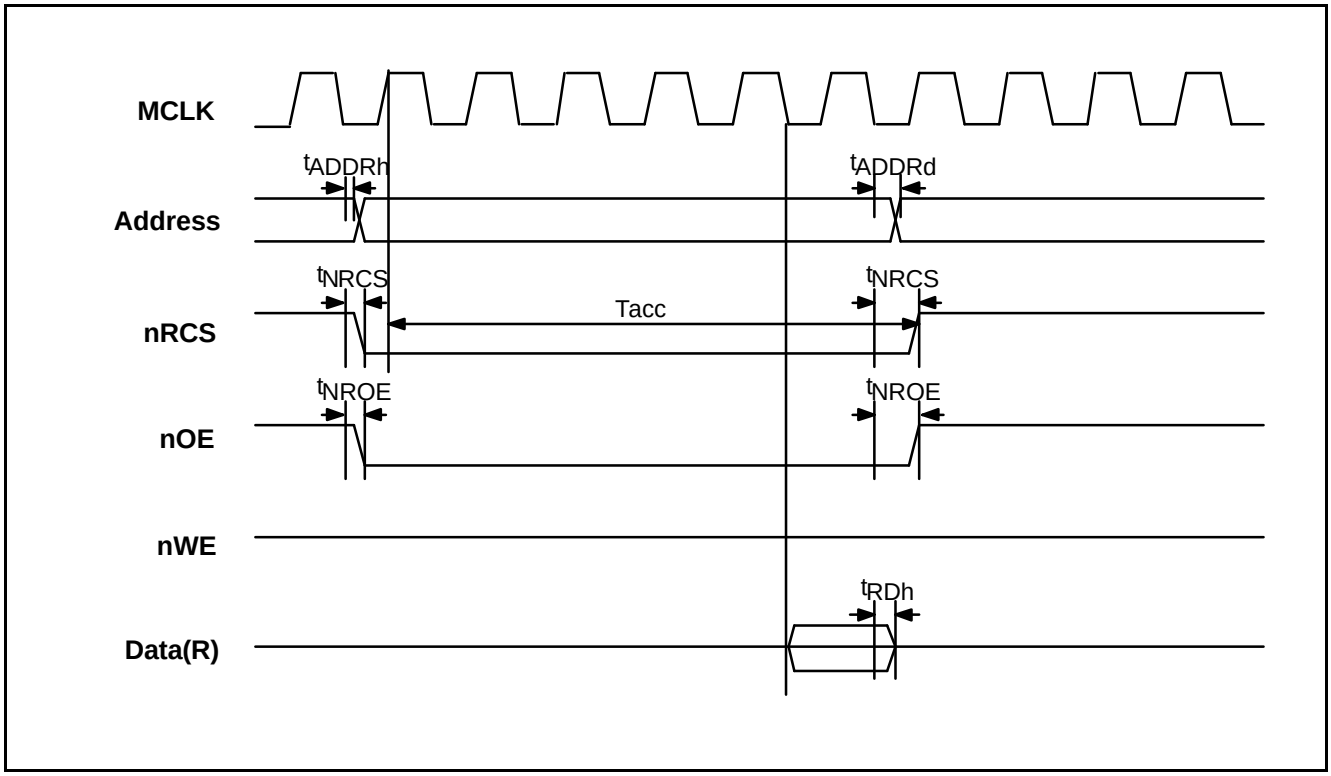


Figure 4-12. SRAM Read Timing

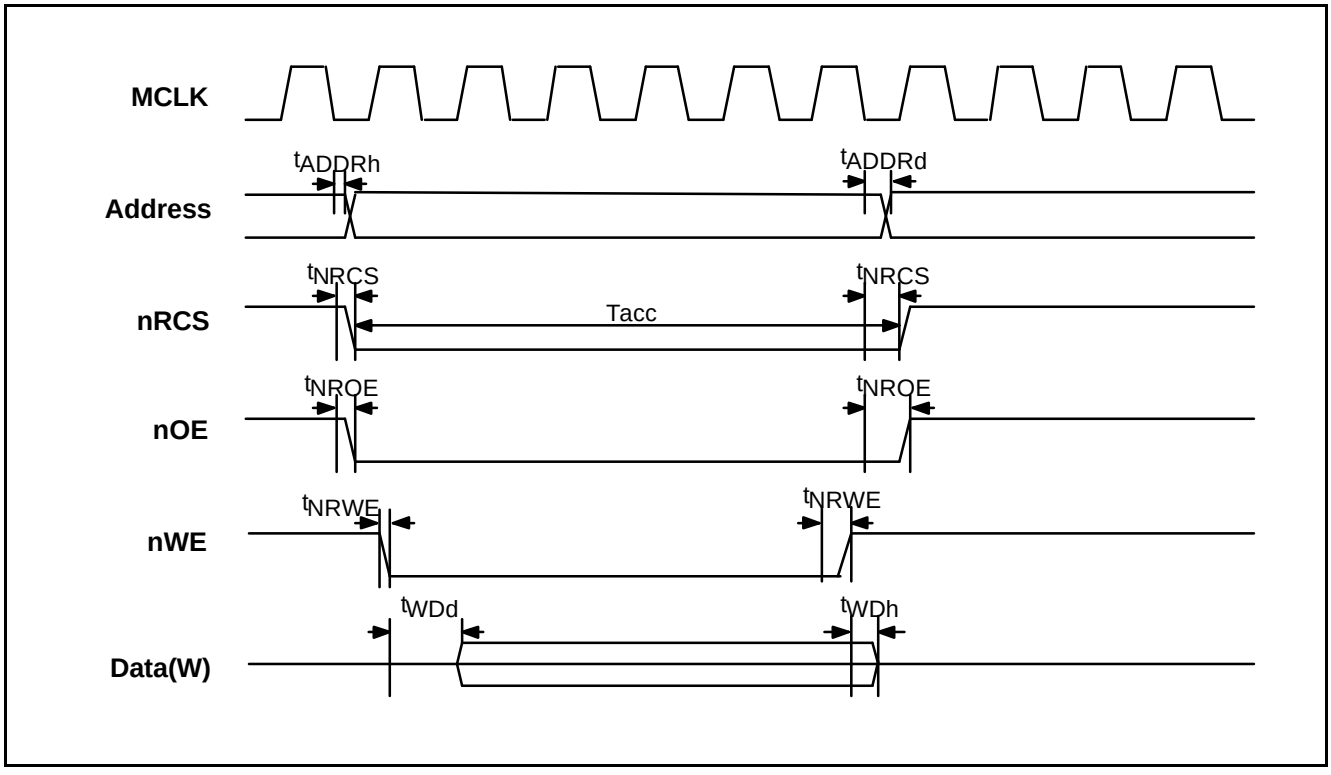


Figure 4-13. SRAM Write Timing

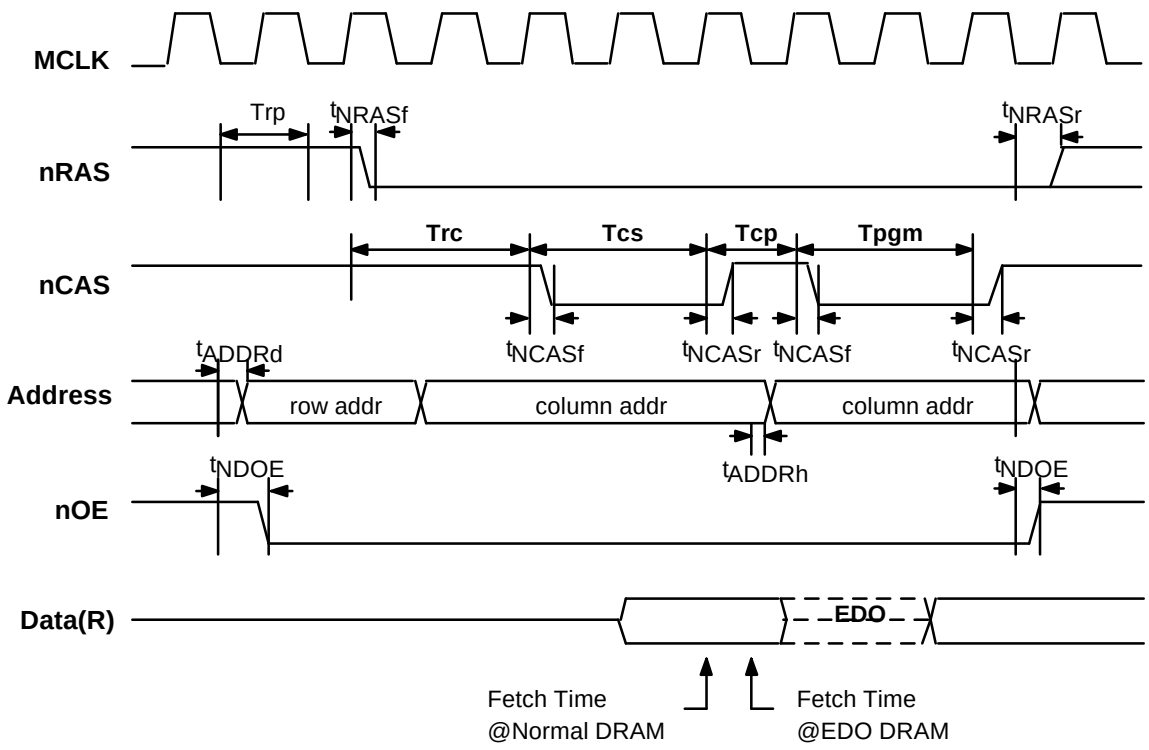


Figure 4-14. DRAM Bank Read Timing (Page Mode)

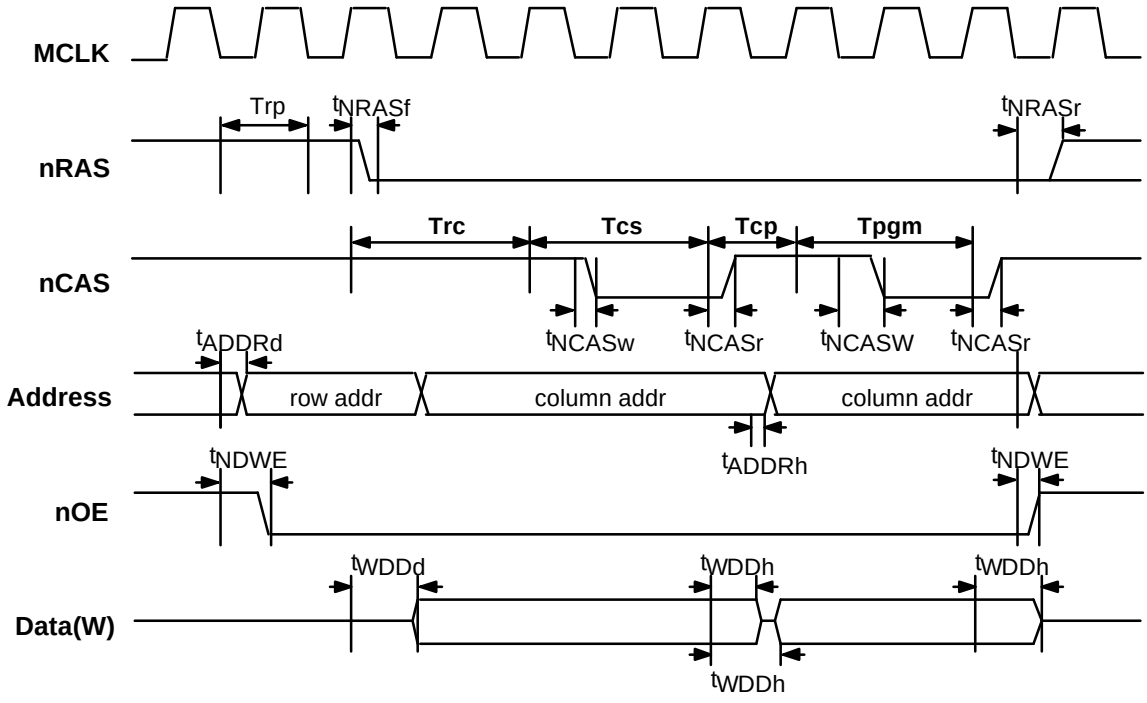


Figure 4-15. DRAM Bank Write Timing (Page Mode)

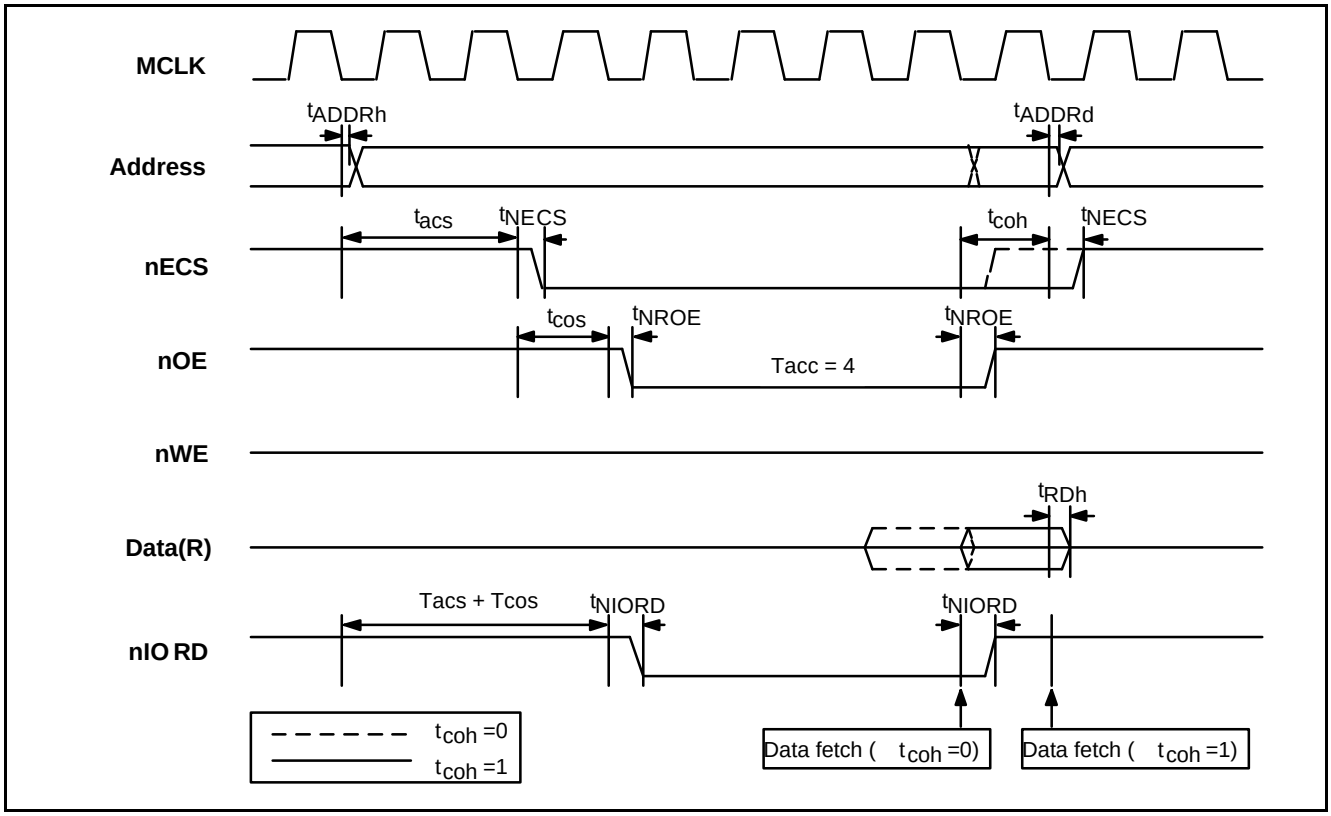


Figure 4-16. Extra - I/O Read Timing (Tcoh = 1, Tacc = 4, Tcos = 1, Tacs = 2)

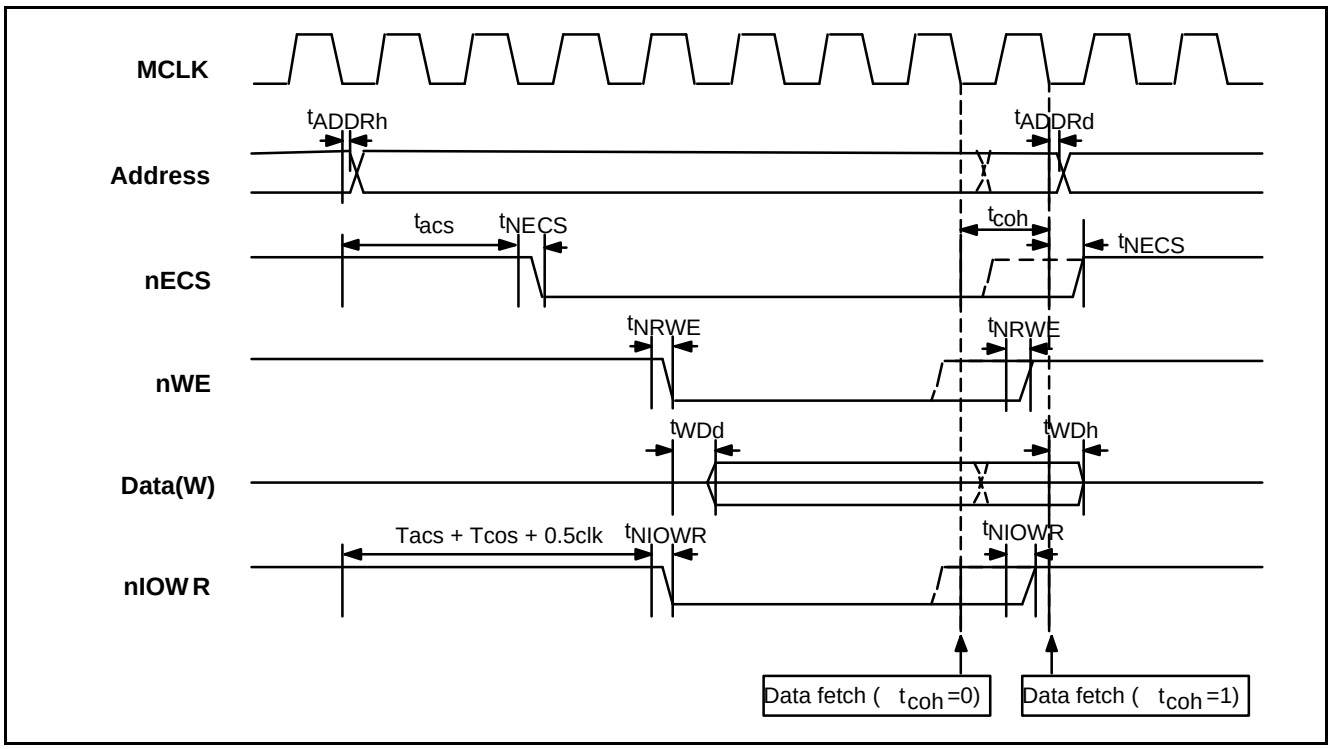


Figure 4-17. Extra - I/O Write Timing

A.C. CHARACTERISTICS

Table 4-2. A.C. Timing Characteristics(Ta = 0°C to +70°C, VDD = 4.75V to 5.25V)

Name	Description	Min.	Max.	Unit
tADDRh	Address Hold Time	7.0		ns
tADDRd	Address Delay Time		25.1	ns
tNRCS	ROM Bank Chip Select Delay Time		20.6	ns
tNROE	ROM/SRAM/ExtIO Bank Out Enable Delay		23.5	ns
tNRWE	SRAM or ExtIO Bank Write Enable Delay		18.2	ns
tRDh	Read Data Hold Time	3.0		ns
tWDd	Write Data Delay Time(SRAM/ExtIO)		9.8	ns
tWDh	Write Data Hold Time(SRAM/ExtIO)	26.3		ns
tNRASf	DRAM Raw Address Strobe Active Delay		15.2	ns
tNRASr	DRAM Raw Address Strobe Release Delay		27.0	ns
tNCASf	DRAM Column Address Strobe Active Delay		16.1	ns
tNCASr	DRAM CAS Signal Release Delay Time		17.1	ns
tNCASw	DRAM CAS Write Active Delay		19.8	ns
tNDWE	DRAM Bank Write Enable Delay Time		24.4	ns
tNDOE	DRAM Bank Out Enable Delay Time		23.5	ns
tNECS	External IO Bank Chip Select Delay Time		20.6	ns
tNIORD	Special IO Bank Read Signal Delay Time		23.5	ns
tNIOWR	Special IO Bank Write Signal Delay Time		18.2	ns
tWDDd	DRAM Write Data Delay Time(DRAM)		14.2	ns
tWDDh	DRAM Write Data Hold Time(DRAM)	7.4		ns

5 UNIFIED (INSTRUCTION/DATA) CACHE

INTRODUCTION

The KS32C6400 CPU has a 4Kbyte internal unified (instruction/data) cache, which adopts associative two-way set architecture with four-word (16 bytes) line size. It has a write-through policy to keep data coherency. When cache miss occurs, four words of memory are fetched sequentially from external memory. It has an LRU (Least Recently Used) algorithm to raise the hit ratio. If it is used 2KB cache, the others 2KB can used internal SRAM.

Typically, RISC CPU uses the instruction and data cache to improve performance. Without cache, bottleneck, which occurs during the instruction and data fetches from the external memory, may seriously degrade performance. The unified cache deals with instruction and data without distinguishing them.

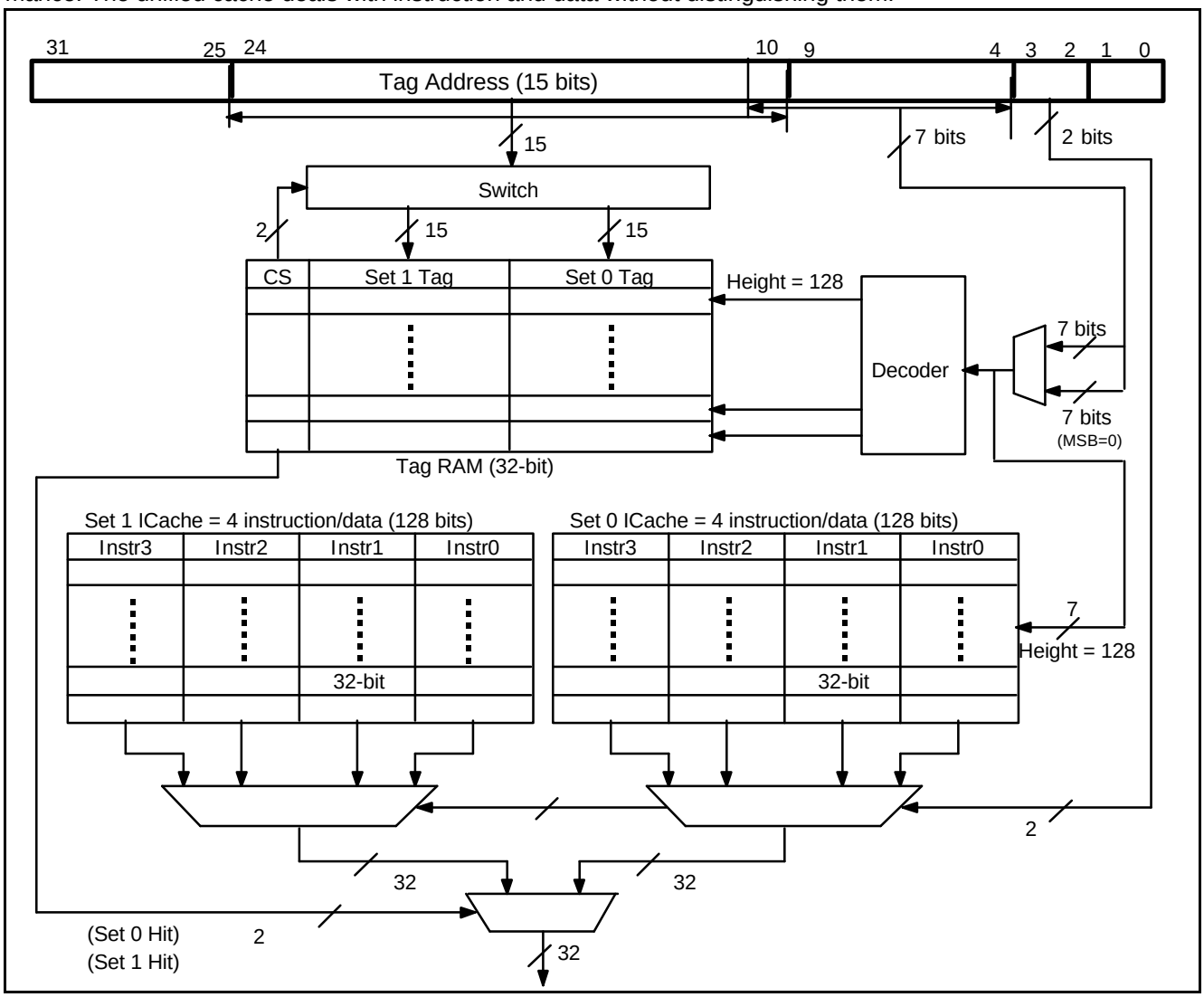


Figure 5-1. Cache Memory Configuration

CACHE OPERATION

CACHE ORGANIZATION

The KS32C6400 cache has a 4Kbyte cache memory and one small Tag RAM. The Tag RAM has a 2-bit CS (Cache Status) and two set Tag memories for set 0 and 1. Each Tag set has a 15-bit address field [24:10] which is stored in the cache memory. The 2-bit CS indicates the validity of cached data of the corresponding cache memory line. It is also used for the cache replacing algorithm and for selecting the data coming from Set 0 and 1. Cache memory has two sets, Set 0 and Set 1. Each set has 128-lines and each line has four words of memory space (128-bits).

CACHE REPLACE OPERATION

After a system is initialized, the value of CS is set to "00", signifying that the contents of set 0 and set 1 cache memory are invalid. When a cache fill occurs, the value of CS is changed to "01" at the specified line, which signifies that only set 0 is valid. When the subsequent cache fill occurs, the value of CS will be "11" at the specified line, which represents that contents of both set 0 and set 1 are valid.

When the contents of the two set are valid and the content replacement is required due to the cache miss, the value of CS is changed to "10" at the specified line, signifying that the content of set 0 is replaced. When the value of CS is "10" and another contents replacement is required due to the cache miss, the content of set 1 will be replaced by changing the value of CS to "11".

Conclusively, at normal steady state, the value of CS will be changed from 11 to 10 (10 to 11), which is information for the implementation of a 2-bit pseudo LRU (Least Recently Used) replacement policy.

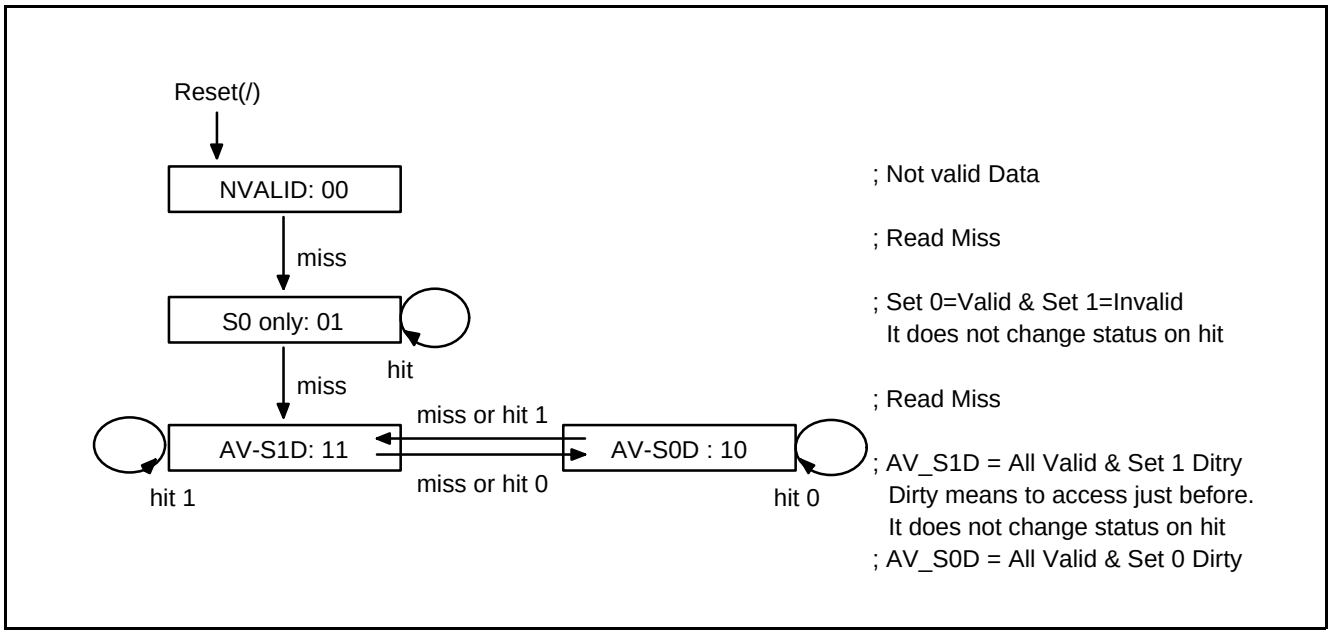


Figure 5-2. CS-Bit Status Diagram

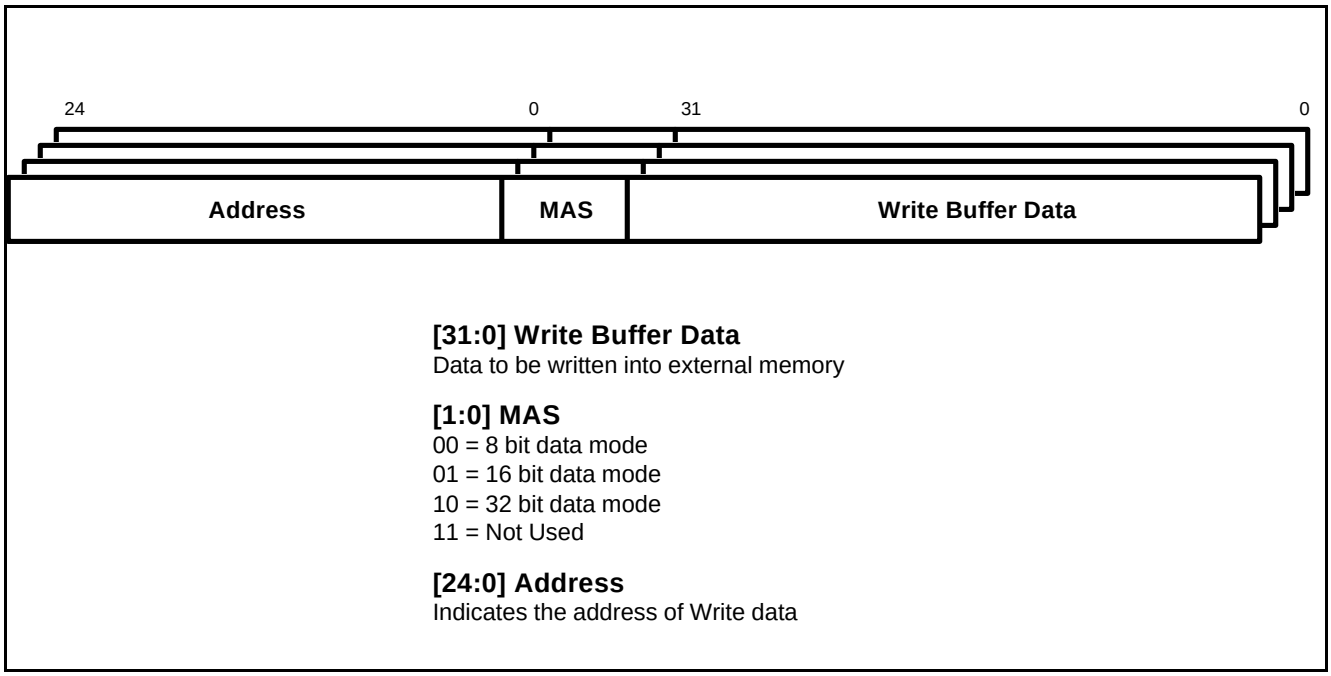
CACHE DISABLE OPERATION

The KS32C6400 cache provides programmable entire-cache-enable/disable mode. You can enable cache by setting the value of CE in SYSCFG to 1, and disable it by clearing SYSCFG[1] to zero. When the cache disable mode is specified, instructions and data are always fetched from external memory. The KS32C6400 can also provide non-cacheable areas in cache-enable mode for some particular memory access operations, such as the DMA operation. The two non-cacheable areas are specified by four special registers to be introduced later.

You have to be cautious about data coherency when the cache memory is enabled again, because the cache memory does not have auto flush mode. You also have to be cautious whether or not DMA changes memory data. The DMA accessible memory area should be non-cacheable to keep the data coherency. To keep data coherency between cache and external memory, the KS32C6400 uses the write-through method.

WRITE BUFFER OPERATION

The KS32C6400 has four Write Buffer Registers to enhance memory writing performance. When Write Buffer mode is enabled, the CPU writes data into the write buffer instead of an external memory when the external bus is already occupied by another bus master like that of DMA. The Write buffer has 4 registers and each register includes a 32-bit data field, a 25-bit address field and a 2-bit status field. The system manager executes all operations of the write buffer



[31:0] Write Buffer Data
Data to be written into external memory

[1:0] MAS
00 = 8 bit data mode
01 = 16 bit data mode
10 = 32 bit data mode
11 = Not Used

[24:0] Address
Indicates the address of Write data

Figure 5-3. Write Buffer Configuration

CACHE CONTROL REGISTERS

The KS32C6400 cache provides two non-cacheable areas. It has four Cache Control registers to specify two non-cacheable areas. Usually a cache stores any data in the whole system memory area, but sometimes it needs a non-cacheable region to keep the data consistency between the external memory and the cache memory.

The KS32C6400 provides two non-cacheable areas and each of them requires two Cache Control Registers to indicate the start and end address of the non-cacheable area. If a non-cacheable area is specified, the area is not cached when read miss occurs.

Register	Offset Address	R/W	Description	Reset Value
CACHNAB0	0x1000	R/W	Start address of non-cacheable area 0	0x00000000
CACHNAE0	0x1400	R/W	End address of non-cacheable area 0 plus one	0x00000000
CACHNAB1	0x1800	R/W	Start address of non-cacheable area 1	0x00000000
CACHNAE1	0x1C00	R/W	End address of non-cacheable area 1 plus one	0x00000000

31	25	24	9	8	7	6	5	4	3	2	1	0
		START/END ADDRESS		0	0	0	0	0	0	0	0	0

[24:9] Non-Cacheable Start/End Address — These 16bit address becomes the upper address of the area.([24:9]), Minimum non-cachable area is 512 byte because offset address is 9 bit.

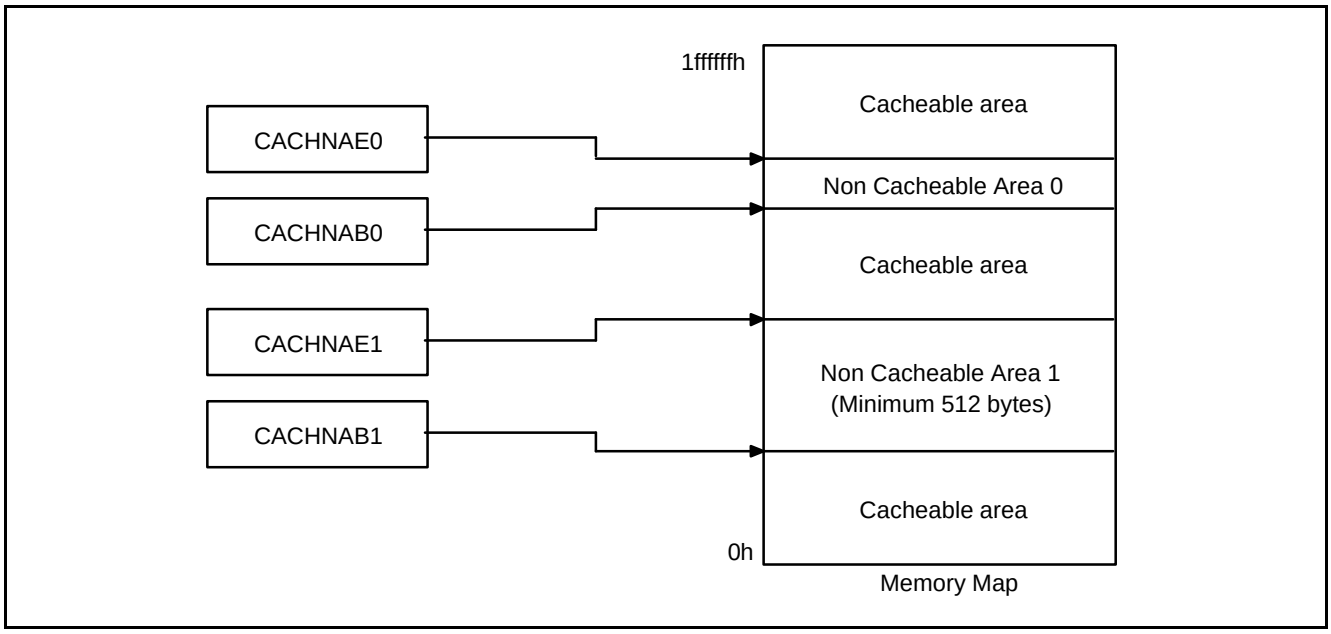


Figure 5-4. Non-Cacheable Area Register

SELECTED CACHE 2K/4K AND INTERNAL SRAM

The KS32C6400 has 2K Internal RAM and 4K Cache that is also used 2K Cache if you disable 4K Cache. If it is used 2K only, you can use the others 2K Cache Memory like Internal RAM.Cache Select bit of SYSCFG must be setted before setting cache enable bit.

Internal SRAM start address is 2800h + start address of SYSCFG and if Cache is setted 2KB, start address of the others Cache 2K range which is used Internal SRAM is 2000h + start address of SYSCFG.when it is used 2KB-Cache/4KB-Internal SRAM, accessible range of Internal RAM is 2000h - 2FFFh. or when it is used 4KB-Cache/2KB-Internal RAM, accessible range of Internal RAM is 2800h - 2FFFh.

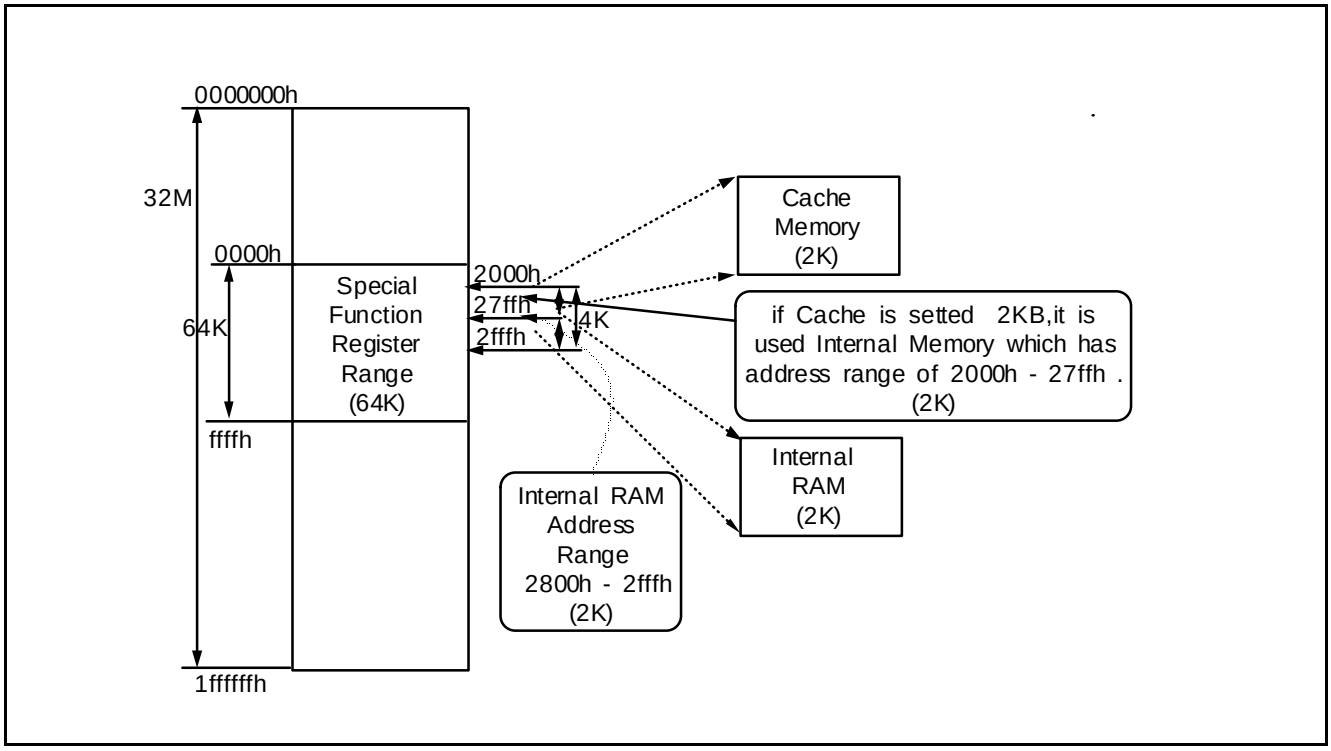


Figure 5-5. Internal SRAM Area

BLACK PAGE

6 DERASTERIZER

INTRODUCTION

The KS32C6400 Derasterizer provides 16 x 16 bit image data rotation feature. The derasterizer consists of 16 registers with 16-bit data width. The 16 x 16-bit register array is used to rotate the raster image data 90 or 270 degree.

Register	Offset Address	R/W	Description	Reset Value
DRAST0	0x00006000	R/W	16-bit derasterizer data register 0	Undef.
DRAST1	0x00006004	R/W	16-bit derasterizer data register 1	Undef.
...	...	...	...	...
DRAST14	0x00006038	R/W	16-bit derasterizer data register 14	Undef.
DRAST15	0x0000603C	R/W	16-bit derasterizer data register 15	Undef.

NOTE: When h[15:0] is written and v[15:0] is read, the address of DRAST0-DRAST15 is used.

ROTATION

To rotate image data, you should fill image data into the 16 x 16 bit register array from DRAST0 to DRAST15, horizontally. The image data, which is made by reading the 16 x 16, has the rotated image. The rotate direction depends on shift control register, SFTCON[3]. When SFTCON[3] is 0, read image data is rotated by 90 degree and when SFTCON[3] is set to 1, the image data is rotated by 270 degree.

Write: h0 –> h15 (DRAST0 –> DRAST15)
Read: 90 degree: (horizontal direction) v0 –> v15 => (vertical direction) MSB –> h15, LSB –> h0
270 degree: (horizontal direction) v15 –> v0 => (vertical direction) MSB –> h0, LSB –> h15

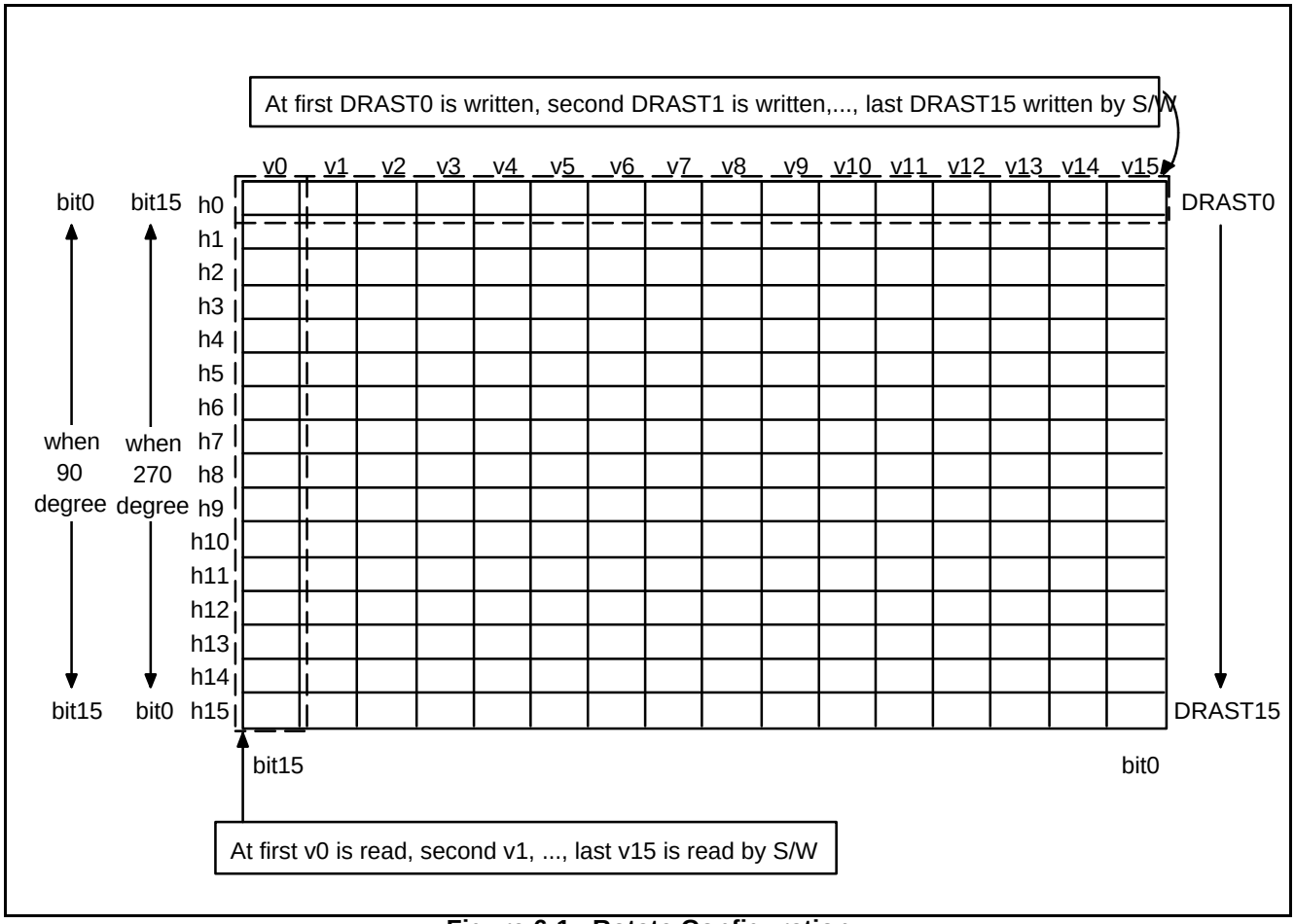


Figure 6-1. Rotate Configuration

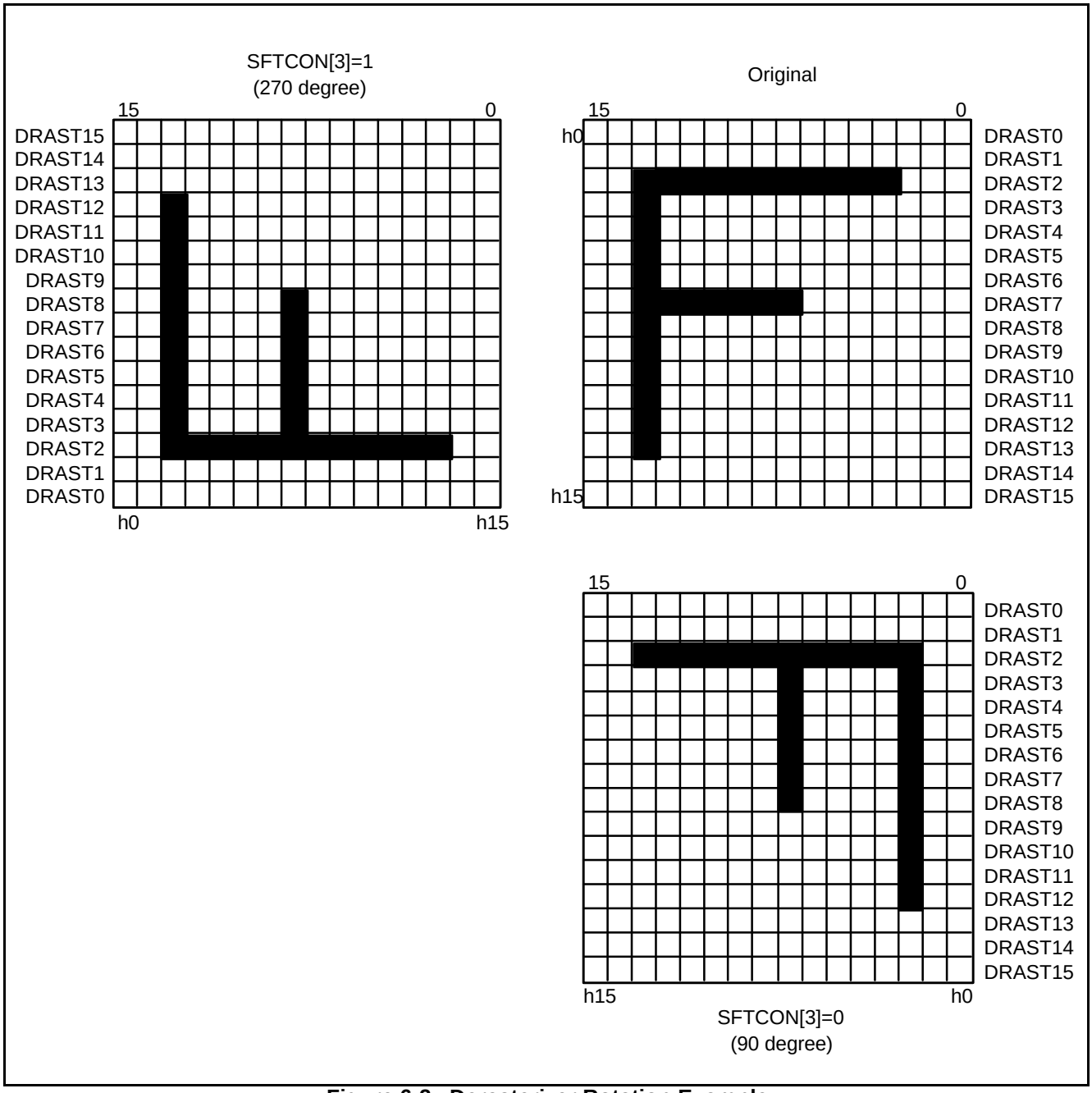


Figure 6-2. Derasterizer Rotation Example

BLANK PAGE

7 SHIFT CONTROL

INTRODUCTION

The KS32C6400 provides data shift and reverse (180 degree rotation) features. Shift module has one 16-bit register for data reverse operation and six 32-bit registers and one 16-bit register for data shift operation. The seven data shift operation registers have seven serially connected registers for 208-bit data shift operation.

DATA REVERSE REGISTER (DATARVS)

The data reverse operation is a kind of data position exchanger. The written data bit stream exchanges its position so that the MSB of the original goes to the LSB of the reversed data and the LSB of the original goes to the MSB of the reversed data.

Normal (MSB-------LSB) => reversed (LSB-------MSB)

Data reverse operation is useful for data rotation operation. The rotator block supports only 90 and 270degree operation. Data reverse operation behaves the same as to the 180 degree rotator operation.

Registers	Offset Address	R/W	Description	Reset Value
DATARVS	0x7000	R/W	Data transfer	0x00000000

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		REVERSE DATA															

[15:0] Reverse Data

DATARVS keeps the reversed bit stream of the data. You can get reversed data right by read operation after writing.
Bit 15 ↔ Bit 00
Bit 14 ↔ Bit 01
Bit 13 ↔ Bit 02
Bit 12 ↔ Bit 03
Bit 11 ↔ Bit 04
Bit 10 ↔ Bit 05
Bit 9 ↔ Bit 06
Bit 8 ↔ Bit 07

NOTES

1. MSB is placed at bit 15 and LSB at bit 0 while writing 16-bit data to this register
2. Read the data from this register; MSB is bit 0 and LSB is bit 15.

Example:

Input ----------> Reverse data register ----------> Output
"1011 0111 0000 1010" "0101 0000 1110 1101"

SHIFT CONTROL REGISTER

Shift Control Register (SFTCON) in the KS32C6400 specifies the rotation degree of the derasterizer data, the direction of data shift, the shift mode, and the status of the shift mode operation (shifting or finished).

Registers	Offset Address	R/W	Description	Reset Value
SFTCON	0x7004	R/W	Shift control register	0x00000004

31	4	3	2	1	0
		X	X	X	X

[0] Shift Mode
0 = Fill with zero
1 = Rotate

[1] Direction of shift
0 = Down(Right)
1 = Up(Left)

[2] Shift Status(read only)
0 = Shifting
1 = Finished

[3] Direction of Derasterzier
0 = 90 degree
1 = 270 degree

SHIFT COUNT REGISTER

Shift Count Register (SFTCNT) specifies the amount of shift operation. When a non-zero value is written into this register, the shifter starts data shifting up to the specific amount. When shift operation has finished, the SFT-CON[2] is set to "1". The execution time is proportional to the amount of shift.

Registers	Offset Address	R/W	Description	Reset Value
SFTCNT	0x7008	R/W	Shift count register	0x00000000

31	8	7	6	5	4	3	2	1	0
		SHIFT COUNTER							

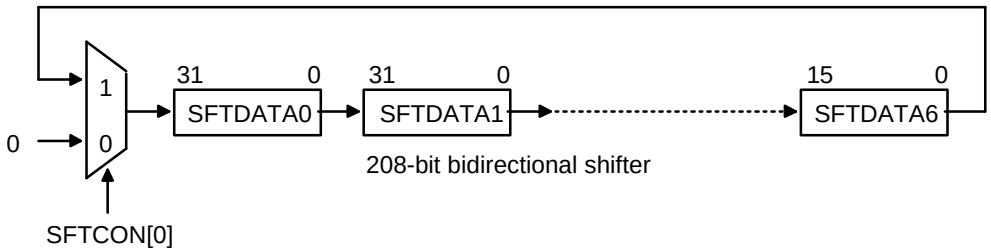
[7:0] Shift Counter
This data specifies the amount of shift count

SHIFT WORD DATA REGISTER

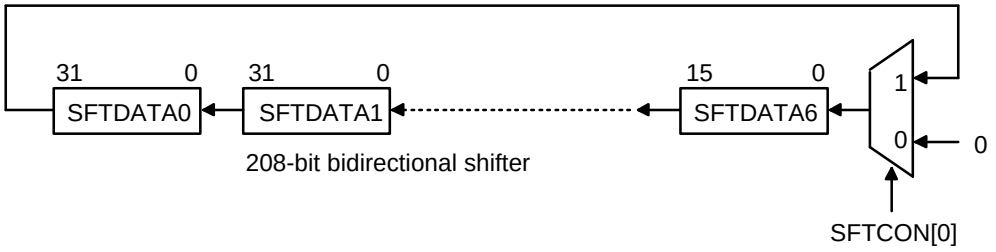
The shift control block in KS32C6400 has seven serially connected Shift Word Data Registers (SFTDATA0-6). These serially connected registers form 208-bit bidirectional shift register block. The shift operation mode can be manipulated by the SFTCON register. The SFTCON[0] bit controls the shift mode such as fill-with-0 mode or rotate (wrap) mode and the SFTCON[1] bit controls shift direction, left or right.

Registers	Offset Address	R/W	Description	Reset Value
SFTDATA0	0x700C	R/W	Shift word data 0 register, 32bits	Undef.
SFTDATA1	0x7010	R/W	Shift word data 1 register, 32bits	Undef.
....	....	....	....	....
SFTDATA5	0x7020	R/W	Shift word data 5 register, 32bits	Undef.
SFTDATA6	0x7024	R/W	Shift word data 6 register, 16bits	Undef.

Example 1: Shift Right Mode (when SFTCON[1]='0')



Example 2: Shift Left Mode (when SFTCON[1]='1')



When programming:

1. Load values into SFTDATA0-SFDATA6 (Shift Word Data Registers)
2. Set the value of SFTCON (Shift Control Register)
3. Set the value of SFTCNT (Shift Counter Register)

BLANK PAGE

8 TIMER

INTRODUCTION

The KS32C6400 has three 16-bit timers. The three timer blocks share an 8-bit prescaler and a clock-divider which has 4 different divided signals. Each timer block receives its own clock signals, for example, the Timer Clock, from the clock divider which receives the clock from the 8-bit prescalar. The 8-bit prescaler is programmable and divides the MCLK signal depending on the loading value which is stored in TSTCON[14:7].
The timer count value register (TBCNTn) has the initial count value which is loaded into the down-counter when the timer is enabled. Each timer has its own 16-bit down-counter which is driven by the timer clock. When the down-counter reaches zero, the timer interrupt request is generated to inform the CPU that the timer operation is completed. When the timer counter reaches zero, the value of corresponding TBCNTn is automatically loaded into the down-counter to continue the next operation. However, if the timer stops, for example, by clearing the timer enable bit of TCON during the timer running mode, the count value of TBCNTn will not be reloaded into the counter.

The timer count value register is used to define the duration of the timer operation, and contains the number of timer clocks needed for one operation duration.

The timer duration is:

Timer_clock = MCLK/(prescale_value + 1)/division_factor (Hz)
Timer_duration = count_value x Timer_clock_period = count_value/Timer_clock

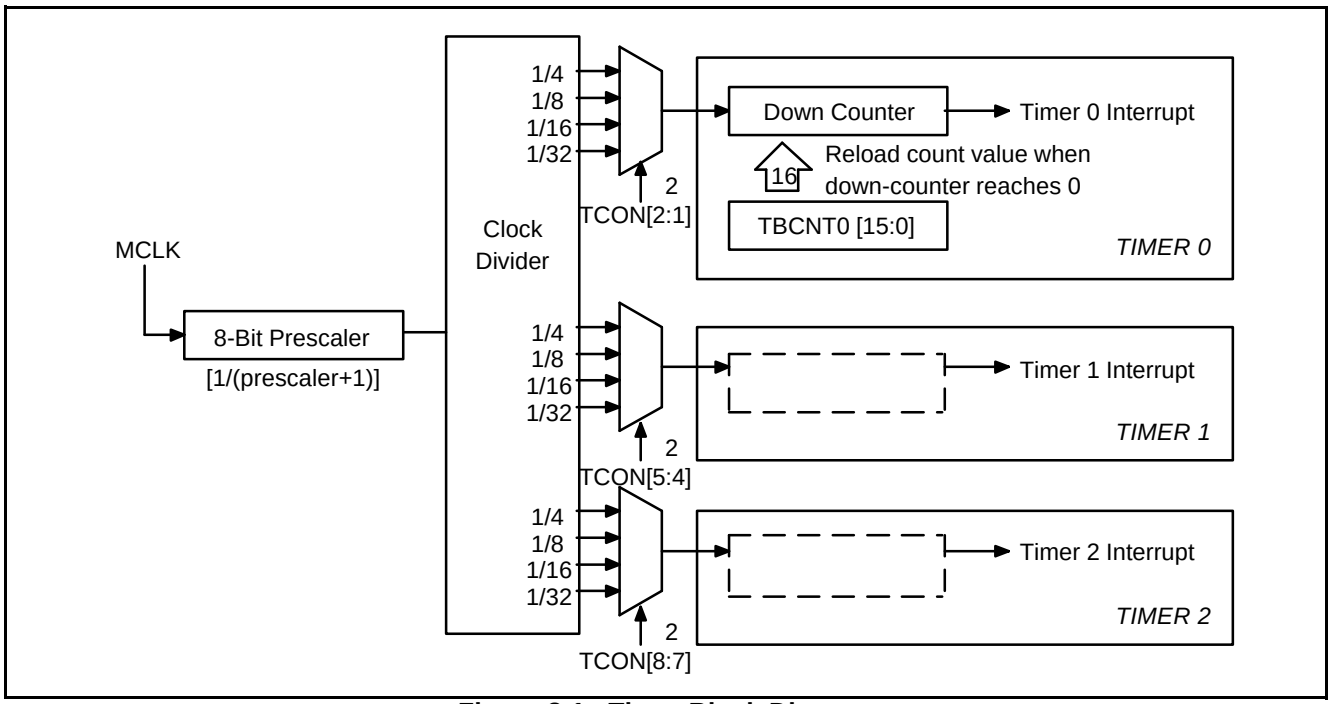


Figure 8-1. Timer Block Diagram

TIMER CONTROL REGISTER

You can disable or enable the timer operation and select the clock-divider output from four-divided signals by using the Timer Control Register (TCON).

Registers	Offset Address	R/W	Description	Reset Value
TCON	0x5800	R/W	System timers control register	0x00000000

31	9	8	7	6	5	4	3	2	1	0
		X		X		X	X		X	X

[0] Timer 0 enable
- 0 = Stop
- 1 = Run

[2:1] Clock division factor selection for timer 0
- 00 = 4
- 01 = 8
- 10 = 16
- 11 = 32

[3] Timer 1 enable
- 0 = Stop
- 1 = Run

[2:1] Clock division factor selection for timer 1
- 00 = 4
- 01 = 8
- 10 = 16
- 11 = 32

[3] Timer 2 enable
- 0 = Stop
- 1 = Run

[2:1] Clock division factor selection for timer 2
- 00 = 4
- 01 = 8
- 10 = 16
- 11 = 32

TIMER COUNT VALUE REGISTER

The timer count value registers, TBCNTn, are used to specify the time-out duration of each timers. The count value will be loaded or reloaded into the down-counter automatically when timer operation is enabled or the down-counter is decreased to zero.

Registers	Offset Address	R/W	Description	Reset Value
TBCNT0	0x5804	R/W	Timer 0 count value register	Undef.
TBCNT1	0x5808	R/W	Timer 1 count value register	Undef.
TBCNT2	0x580C	R/W	Timer 2 count value register	Undef.

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		COUNT VALUE															

[15:0] Timer 0/1/2 Count Value

Although the contents of TBCNTn is changed while the timer is enabled, the timer will keeping counting with previous value untill the timer expired, and load the new base value, start counting again..

Example)

The timer programming sequence is shown below. The count value and timer clock definition, including the prescaling value and clock division factor, should be specified before the timer-enable bit setting.

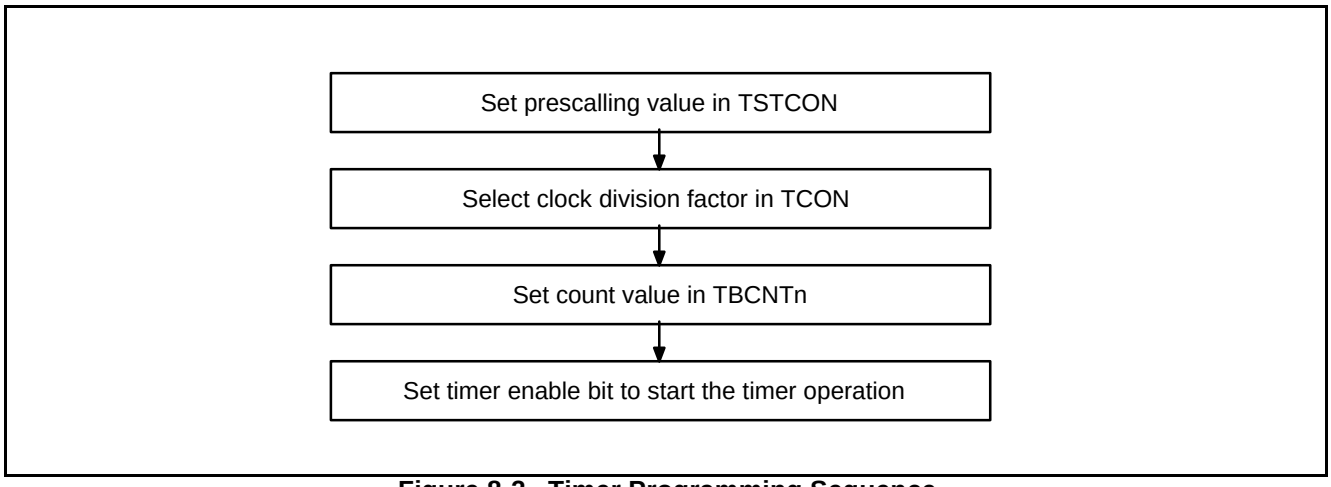

Set prescalling value in TSTCON
Select clock division factor in TCON
Set count value in TBCNTn
Set timer enable bit to start the timer operation

Figure 8-2. Timer Programming Sequence

BLANK PAGE

9 DMA

INTRODUCTION

The KS32C6400 has two general direct memory access channels (DMA0, DMA1) that perform the data transfers between the following sources without CPU intervention:

- Memory and memory
- Parallel port and memory
- Serial port and memory

The on-chip DMA controller can be started by software or by an external DMA request. DMA operation can also be stopped and restarted by software. The CPU can recognize when a DMA operation has been completed by software polling or by a DMA interrupt request. The KS32C6400 DMA controller can increase or decrease source or destination addresses and conduct 8-bit (byte), 16-bit (half-word), or 32-bit (word) data transfers.

Detailed information about the DMA block's operation is provided in the descriptions of the DMA registers.

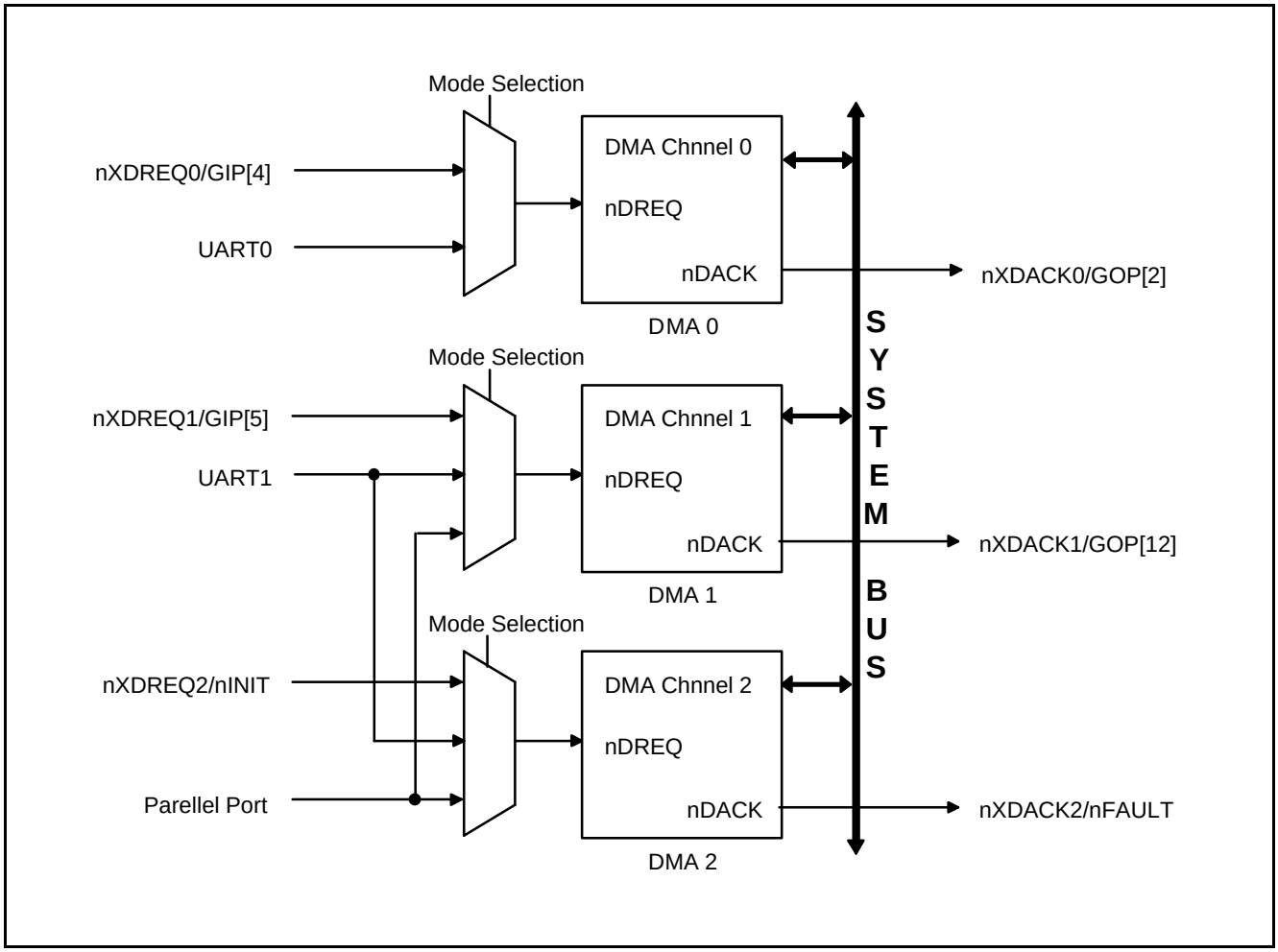


Figure 9-1. DMA0/DMA1 Unit Block Diagram

In case of nXDREQ2/nINIT, should set the IOPMOD[31] to one.

DMA OPERATION

The DMA operation can be summarized as follows:

DMA TRANSFERS

The DMA transfers data directly between a requester and a target. The requester or the target is memory, UART, parallel port, or an external device. An external device requests DMA service by activating the nXDREQ signal.

A channel is programmed by writing to register which contain requester address, target address, amount of data, and other control contents.

UART, parallel port, external I/O, and software (memory) can request DMA service. UART and parallel port are internally connected to the DMA. Especially, UART0 requests the DMA service to DMA0 and UART1 to DMA1, DMA2.

BUS CONTROL ARBITRATION

Because DMA0, DMA1, DMA2 and DRAM controller (DRAM refresh) can request bus control, the bus control priority must be arbitrated. The priority of these bus masters is fixed as follows:

Bus Master Type	Priority
DMA0	1
DRAM controller (DRAM refresh)	2
DMA1	3
DMA2	4

For fast response to the DMA0 request, the DMA0 has the highest priority. Because the DMA0 has higher priority than the DRAM controller, the DMA0 have to be used very carefully to avoid disturbing the DRAM controller to refresh the DRAM. You may think that the DMA0 can not move the large data of DRAM because of the DRAM refresh. But, the DMA0 can transfer the large DRAM data if you do not use continuous DMA0 mode. The DMA0, which doesn't use the continuous mode, releases the internal bus request in a short time after one unit of data is transferred (one word, one half-word (16-bit) or one byte). Just after the bus is released, the DRAM controller can control the bus and refresh DRAMs.

If the DMA2, which has the lower priority than the DRAM controller, holds bus by the continuous mode, the DRAM refresh controller can not control the bus until the DMA2 frees the bus control.

STARTING/ENDING DMA TRANSFERS

DMA starts to transfer data after the DMA receives the service request from the nXDREQ signal, UART, parallel port, or software. When the entire data buffer has been transferred, the DMA becomes idle. If you want to preform another buffer transfer, the DMA must be reprogrammed. When the same buffer transfer is preformed again, the DMA must be reprogrammed.

THE MAJOR DIFFERENCE BETWEEN DMA0, DMA1, DMA2

DMA0, DMA1, and DMA2 has difference as the table 9-1.

Table 1-1. The Major Difference between DMA0, DMA1, and DMA2

Functions	DMA0	DMA1	DMA2
Single mode	O	O	O
Block mode	O	O	O
Demand mode	O	O	X
Fly-by mode	O	O	X

DATA TRANSFER MODE

SINGLE MODE

The DMA request (nXDREQ, UART or PPIC) causes one byte, one half-word, or one word to be transmitted. The single mode requires the DMA request for every data transfer. The nXDREQ signal may be deasserted after checking whether or not the nXDACK is asserted.

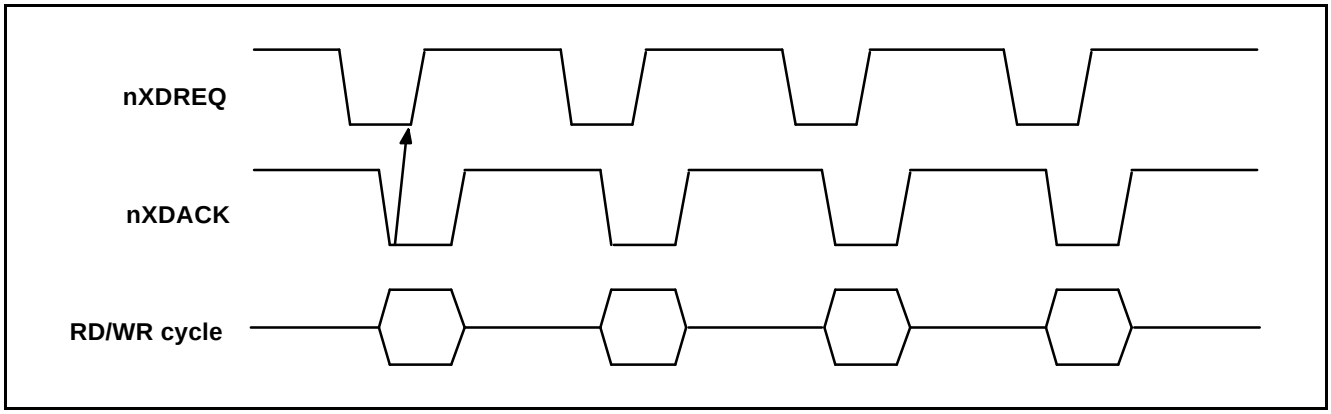


Figure 9-2. External DMA Requests (Single Mode)

BLOCK MODE

The assertion of only one DMA request (nXDREQ, UART, PPIC or S/W) causes the entire data, which is set in control registers, to be transmitted. The DMA transfer will be completed when the counter reaches zero. The nXDREQ signal may be deasserted after checking whether or not nXDACK is transmitted.

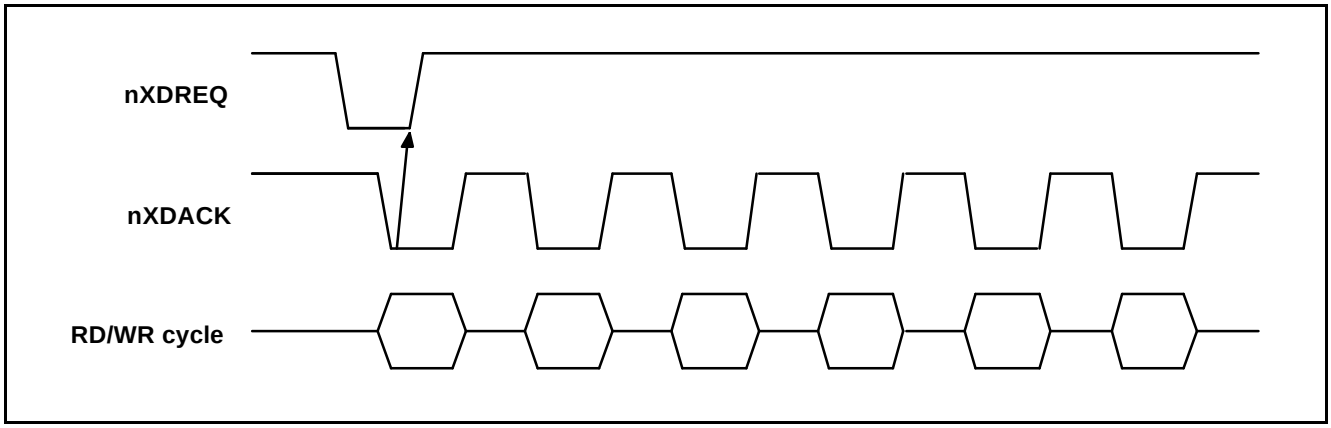


Figure 9-3. External DMA Requests (Block Mode)

DEMAND MODE

The amount of data that DMA transfers depends on how long the DMA request input (nXDREQ) is held active. In demand mode, the DMA(DMA0, DMA1 only) continues to transfer data while the DMA request input (nXDREQ) is held active.

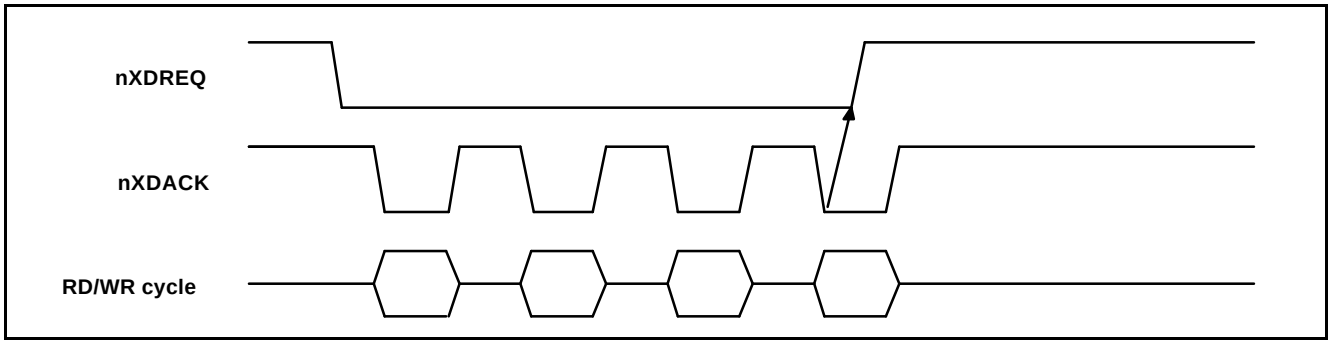


Figure 9-4. External DMA Requests (Demand Mode)

FLY-BY MODE

The amount of data that DMA transfers depends on how long the DMA request input (nXDREQ) is held active. In demand mode, the DMA(DMA0, DMA1 only) continues to transfer data while the DMA request input (nXDREQ) is held active. The Fly-by mode is supported only when EDO-type DRAM with Tcp = 1(2cycle)case is used for memory devices.

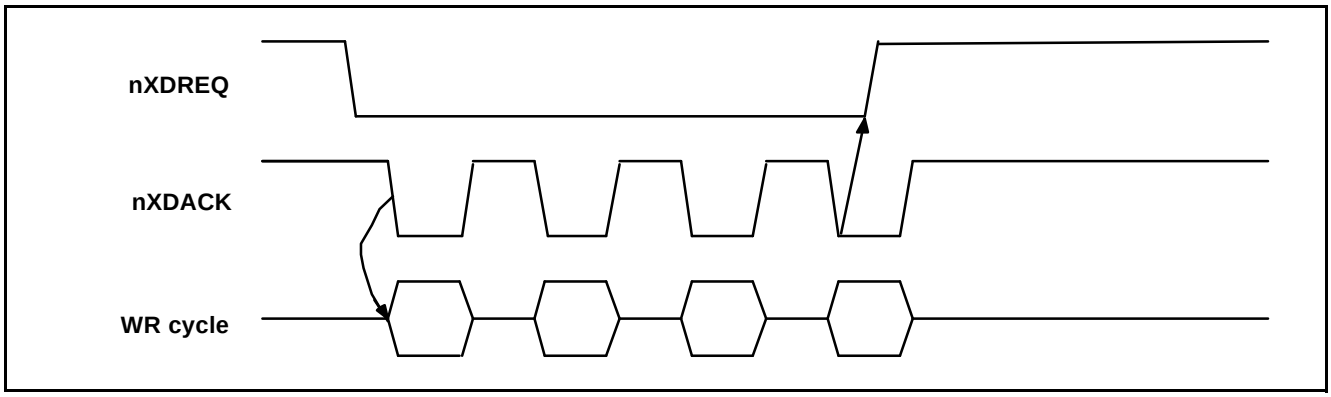


Figure 9-5. External DMA Requests (Fly-by Mode Write Cycle)

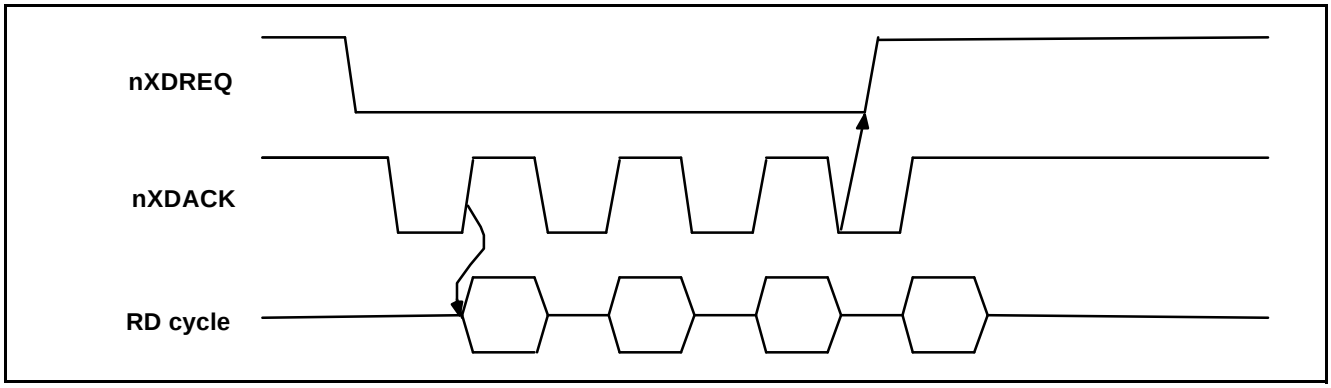


Figure 9-6. External DMA Requests (Fly-by Mode Read Cycle)

DMA0, DMA1 CONTROL REGISTER

Register	Offset Address	R/W	Description	Reset Value
DMACON0	0xC000	R/W	DMA0 control register	0x00000000
DMACON1	0xD800	R/W	DMA1 control register	0x00000000

31	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		x	x				DM	CN	TW		SB	TD	0	SI	SF	DF	SD	DD	MODE		BS	RE

[0] Run enable/disable

Run enable/disable control bit of DMA. To control only this bit, in case of DMA0, use the address 0xC020. in case of DMA1, use the address 0xD820. By using the addresses, you can avoid affecting the other values of the control register.

0 = Disable DMA operation
1 = Enable DMA operation

[1] BUSY status

When the DMA starts, this read-only status bit is automatically set to one. When the DMA is in an idle state, this bit is set to zero.

0 = Idle DMA operation
1 = Active DMA operation

[3:2] Mode Selection

Four sources can initiate DMA operation: software (memory to memory), an external DMA request(nXDREQ) the parallel port, and the UART block. The mode selection bits determine which source can initiate a DMA operation at any given time.

For DMA0	**For DMA1**
00 = Software	00 = Software
01 = External nXDREQ0	01 = External nXDREQ1
10 = Not used	10 = parallel port
11 = UART0 port	11 = UART1 port

[4] Destination address direction

This bit determines whether the destination address will be decreased or increased during a DMA operation.

0 = Increase address
1 = Decrease address

[5] Source address direction

This bit determines whether the source address will be decreased or increased during a DMA operation.

0 = Increase address
1 = Decrease address

[6] Destination address fix

This bit determines whether the destination address will be changed during a DMA operation. You can use this bit to transfer data from multiple sources to a single destination.

0 = Increase/Decrease destination address
1 = Do not change destination address(fix)

[7] Source address fix

This bit determines whether the source address will be changed during a DMA operation. You can use this bit to transfer data from a single source to multiple destinations.

0 = Increase/Decrease destination address
1 = Do not change destination address(fix)

[8] Stop interrupt enable	A DMA operation is started by setting the run enable bit to one and is stopped by clearing the run disable bit to zero. The interrupt, which is generated when the DMA counter is expired, can not be masked by this bit. 0 = Do not generate the stop interrupt when DMA stops 1 = Generate the stop interrupt when DMA stops
[9] Reserved	If this bit should be reserved as zero.
[10] Peripheral direction	When the mode bit is set to '10' (parallel port from/to memory) or '11' (UART from/to memory), this direction bit specifies the direction of the DMA operation. 0 = Paralle/UART to memory 1 = Memory to parallel/UART
[11] Single/block mode	This bit determines the number of external DMA requests(nXDREQ) that are required for the DMA operation. At single mode, the KS32C6400 requires an external DMA request for every DMA operation. At block mode, the KS32C6400 requires only one DMA request during the entire DMA operation. An entire DMA operation is the DMA operation before the value of counter becomes zero. 0 = nXDREQ initiates a single DMA operation 1 = nXDREQ initiates a block DMA operation
[13:12] Transfer width	This determines the transfer data width: byte(8-bit), half-word(16-bit), or word (32-bit). If the transfer data width is a byte, source/destination address will be increased/decreased by one. If it is a half-word, the address will be increased/ decreased by two. If it is a word, the address will be increased/decreased by four. Note that the "transfer width" is not the physical size of data bus. The physical size of data bus is determined by SMR(System Manager Registers) configurations. 00 = Byte(8bit) 01 = Half-word(16bit) 10 = Word(32bit) 11 = Not used
[14] Continuous mode	This bit specifies whether or not the DMA0/1 operation holds the system bus until the count value is changed to zero. Therefore, this bit must be carefully used so that the whole operation time does not exceed appropriate intervals such as DRAM Refresh. 0 = Normal operation 1 = Hold system bus until the whole DMA operation stops.
[15] Demand mode	To speed up the external DMA operation, set this bit. If this bit has been set during the DMA operation, the DMA never goes to the idle state. The external device can control the amount of data transferred/received by hardware. The amount of data is depends on how long nXDREQ signal is active. 0 = Normal external DMA mode 1 = Demand mode
[19] X2M mode	To run the external DMA operation in fly-by mode, should set this bit to one. In this case, external I/O device writes data to memory at the negative edge of nXDACK. Simultaneously, should set this bit, [15] bit both. 0 = Normal mode 1 = Fly-by mode(External I/O to Memory)

[20] M2X mode

To run the external DMA operation in fly-by mode, should set this bit to one. In this case, external I/O device read data to memory at the positive edge of nXDACK. Simultaneously, should set this bit, [15] bit both.

0 = Normal mode
1 = Fly-by mode(External I/O to Memory)

NOTES
1. All control bits have to be configured independently and carefully.
2. Fly-by mode is limited to EDO type DRAM with 2 cycle Tcp(CAS precharge time).

DMA0, DMA1 SOURCE/DESTINATION ADDRESS REGISTER

These registers contain the 25-bit source/destination address of a DMA0, DMA1 channel. Depending on the setting of the DMA0, DMA1 control register(DMACON0, DMACON1), these addresses will be increased/ decreased or will remain the same.

Register	Offset Address	R/W	Description	Reset Value
DMASRC0	0xC004	R/W	DMA0 source address register	Undef.
DMADST0	0xC008	R/W	DMA0 destination address register	Undef.
DMASRC1	0xD804	R/W	DMA1 source address register	Undef.
DMADST1	0xD808	R/W	DMA1 destination address register	Undef.

31 ~ 25	24 ~ 0
	Source/Destination Address

[24:0] Source/Destination address

DMA0, DMA1 TRANSFER COUNT REGISTER

This register contains the 24-bit value which is the number of DMA0, DMA1 transfers. This value is decreased by 1 when one DMA0, DMA1 operation is completed regardless of the width of the data which was transferred.

Register	Offset Address	R/W	Description	Reset Value
DMACNT0	0xC00C	R/W	DMA0 transfer count register	Undef.
DMACNT1	0xD80C	R/W	DMA1 transfer count register	Undef.

31 ~ 24	23 ~ 0
	Number of Transfers

[23:0] Number of Transfers

DMA2 CONTROL REGISTER

DMA2(CDMA) is second DMA. DMA2 can transfer the data by byte swap mode. The UART1 can transfer data through only DMA2.

Register	Offset Address	R/W	Description	Reset Value
DMACON2	0xC800	R/W	DMA2 control register	0x08000000

31 28	27	26 25 24 23 22 21	20	19 18 17 16	15	14	13 12	11	10	9	8	7	6	5	4	3 2	1	0
	x	BUFCNT	x	CDM		0	CN	TW	SB	TD	0	SI	SF	DF	SD	DD	MODE	BS	RE

[0] Run Enable/Disable

Run enable/disable control bit of DMA2. To stop DMA2, you must clear this bit to zero. To control this bit only, use the address 0xC810. By using the 0xC810 address, the other values in the control register will not be affected.

0 = Disable DMA2 operation
1 = Enable DMA2 operation

[1] BUSY Status

When the DMA2 starts, this read-only status bit is automatically set to one. When it is zero, the DMA2 is in an idle state.

0 = Idle DMA2 operation
1 = Active DMA2 operation

[3:2] Mode Selection

Four sources can initiate DMA2 operation: software(memory to memory), an external DMA2 request(nXDREQ2), the parallel port, and the UART block. The mode selection bits determine which source can initiate a DMA2 operation at any given time.

00 = Software 01 = External nXDREQ2
10 = Parallel port 11 = UART1 port

[4] Destination address direction

This bit determines whether the destination address will be decreased or increased during a DMA2 operation.

0 = Increase address
1 = Decrease address

[5] Source address direction

This bit determines whether the source address will be decreased or increased during a DMA2 operation.

0 = Increase address
1 = Decrease address

[6] Destination address fix

This bit determines whether the destination address will or will not change during a DMA2 operation. This feature is used when transferring data from multiple sources to a single destination.

0 = Increase/Decrease destination address
1 = Do not change destination address(fix)

[7] Source address fix

This bit determines whether or not the source address will be change during a DMA2 operation. This feature is used when transferring data from multiple sources to a single destination.

0 = Increase/Decrease destination address
1 = Do not change destination address(fix)

[8] Stop interrupt enable

A DMA2 operation is started by setting the run enable bit to one and is stopped by clearing the disable bit to zero. The interrupt, which is generated when the DMA2 counter is expired, can not be masked by this bit.

0 = Do not generate the stop interrupt when DMA2 stops
1 = Generate the stop interrupt when DMA2 stops

[9] Reserved

If this bit should be reserved as zero.

[10] Peripheral direction

When the mode bit is set to '10' (parallel port from / to memory) or '11' (UART from / to memory), this direction bit specifies the direction of the DMA2 operation. If this bit is set to '1', the DMA2 operates from memory to peripheral (parallel port/UART). If this bit is cleared to '0', the DMA2 operates from peripheral to memory.

0 = Paralle/UART to memory
1 = Memory to parallel/UART

[11] Single/block mode

This bit determines the number of external DMA2 requests(nXDREQ2) that are required for the DMA operation. At single mode, the KS32C6400 requires an external DMA2 request for every DMA2 operation. At block mode, the KS32C6400 requires only one DMA2 request during the entire DMA2 operation. An entire DMA2 operation is the DMA2 operation before the value of counter becomes zero.

0 = Single mode
1 = Block mode

[13:12] Transfer width

This determines the transfer data width: byte(8-bit), half-word(16-bit), or word (32-bit). If the transfer data width is a byte, source/destination address will be increased/decreased by one. If it is a half-word, the address will be increased/ decreased by two. If a word, the address changes by four. Note that the "transfer width" is not the size of the phisical data bus. The size of physical data bus is determined by SMR (System Manager Registers) configurations.

00 = Byte(8bit) 01 = Half-word(16bit)
10 = Word(32bit) 11 = Not used

[14] Continuous mode

This bit specifies that the DMA2 operation hold the system bus until the count value to set to zero. Therefore, this bit must be carefully used unless the whole operation time can not exceed appropriate interval such as DRAM Refresh.

0 = Normal operation
1 = Hold system bus until the whole DMA operation stops.

[15] Burst mode

If this bit is set to one, DMA2 operates in 4 words burst mode
Memory to memory: 4 words read --> 4 words write
External to memory: 4 words read --> 4 words write
Memory to peripheral: 4 words read --> 16 bytes write
Peripheral to memory: 16 bytes read --> 4 words write
In memory to memory operation, the DMA2CNT value is decreased by one when each 4-word read or write operation is completed; while in memory to/ from peripheral operations, the DMA2CNT value is decreased by one when each one byte read or write operation is completed.

0 = Normal operation
1 = Burst mode operation.

[20:19] Compress/Decompress mode

This bit-pair determines whether a compression/decompression operation will be carried out during a data transfer. Please note that you cannot use compression/decompression mode with the burst mode together. Normally, the compression/decompression operation can only be carried out in a DMA2 operation requested by software (that is, the mode bit-pair[3:2] is set to "00"), and is used only for memory-to-memory transfers in byte units.

00 = Normal 01 = Decompress
10 = Compress 11 = Not used

[21] Compress flush

In compression mode, you set this bit to flush the compressed data remaining in the DMA2 internal buffer to memory when the DMA2CNT value reachec to zero. When the flush operation is finished, the DMA2CON[27] bit (the idle flag) is set automatically. Please, note that the compression operation must be completed before you set this bit. To initialize the next DMA2 operation, you should clear this bit.

0 = No operation
1 = Flush remaining compressed data to memory

[26:22] Buffer count

When DMA2 is operating in burst mode, this read-only count value indicates the number of data bytes remaining in DMA2 buffer when the burst data transfer is completed. In other words, it indicates the number of MOD16(n), where "n" is the total number of bytes of data to be transferred that you defined before the burst DMA operation starts.
The internal 16-byte DMA2 buffer, CDMABUF, corresponds the address offset range, 0xC820 to 0xC82F. Therefore, to obtain the remaining data bytes when the burst data transfer is completed, you must access these memory addresses.
For example, if the buffer count is 3, the remaining data in 0xC820, 0xC821, and 0xC822 should be transferred by software in the last step of the burst mode transfer.

[27] Idle Flag

This read-only bit indicates whether or not DMA2 is in an idle state. If you issue a compress flush oiperation, you must check the value of this bit until it turns to be "1" before you start to execute the next instruction, because the next instruction can be normally executed only when the DMA2 is in an idle state.

0 = Idle DMA2.
1 = Active DMA2.

NOTES

1. All control bits have to be configured independently and carefully.

DMA2 SOURCE/DESTINATION ADDRESS REGISTER

These registers contain the 25-bit source/destination address of a DMA2 channel. Depending on the setting of the DMA2 control register(DMACON2), these addresses will be increased/decreased or will remain the same.

Register	Offset Address	R/W	Description	Reset Value
DMASRC2	0xC804	R/W	DMA2 source address register	Undef.
DMADST2	0xC808	R/W	DMA2 destination address register	Undef.

31 25	24 0
	Source/Destination Address

[24:0] Source/Destination address

DMA2 TRANSFER COUNT REGISTER

This register contains the 24-bit value which is the number of DMA2 transfers. This value is decreased by 1 when one DMA2 operation is completed regardless of the width of the data which was transferred.

Register	Offset Address	R/W	Description	Reset Value
DMACNT2	0xC80C	R/W	DMA2 transfer count register	Undef.

31 24	23 0
	Number of Transfers

[23:0] Number of Transfers

10 PARALLEL PORT INTERFACE

INTRODUCTION

The KS32C6400's parallel port interface controller(PPIC) supports four IEEE Standard 1284 communication modes:

- Compatibility mode (Centronics TM)
- Nibble mode
- Byte mode
- Enhanced Capabilities Port (ECP) mode

The PPIC also supports all variants of these communication modes, including device ID requests and run-length encoded (RLE) data compression.

The PPIC contains specific hardware to support the following operations:

- Automatic hardware handshaking between host and peripheral compatible with ECP modes
- Run-length detection and compression/decompression data between host and peripheral during ECP mode transfers

These features can substantially improve data transfer rates when the KS32C6400 operates the parallel port in the Compatibility or ECP mode.
In addition, hardware handshaking over the parallel port can be enabled or disabled by software. This gives you the direct control of PPIC signals as well as the eventual use of future protocols. Other operations defined in the IEEE Standard 1284, such as negotiation, Nibble mode and Byte mode data transfers, and termination cycles, must be carried out by software. The IEEE 1284 EPP communications mode is not supported.

NOTE

Here we assume that you are familiar with the parallel port communication protocols specified in the IEEE 1284 Parallel Port Standard. If you are not, we strongly recommend for you to read this standard beforehand. It would be helpful for you in understanding the contents described in this section.

A detailed technical introduction to the IEEE 1284 Parallel Port Standard can be found in the Web site: http://www.fapo.com/ieee1284.htm

PPIC OPERATING MODES

The KS32C6400 PPIC supports four kinds of handshaking modes for data transfers:

- Software handshaking mode to forward and reverse data transfers
- Compatibility hardware handshaking mode to forward data transfers
- ECP hardware handshaking without RLE support (ECP-without-RLE) mode to forward and reverse data transfers
- ECP hardware handshaking with RLE support (ECP-with-RLE) mode to forward and reverse data transfers

Mode selection is specified in the PPIC control register (PPCON). By setting the PPCON[3:2], one of these four modes is enabled.

SOFTWARE HANDSHAKING MODE

This mode is enabled by setting the PPCON's mode-selection bits, PPCON[3:2], to "00." In this mode, you can use PPIC interrupt event registers (PPINTEN and PPINTPND) and the read/write PPIC status register (PPSTAT) to detect and control the logic levels on all parallel port signal pins. Software can control all parallel port operations, including all four kinds of parallel port communications protocols supported by the KS32C6400 (refer to IEEE 1284 standard for operation control). In addition, it also gives software the flexibility of adopting new and revised protocols.

COMPATIBILITY HARDWARE HANDSHAKING MODE

Compatibility hardware handshaking mode is enabled by setting the PPCON's mode-selection bits as "01", i.e. PPCON[3:2] = 01. In this mode, hardware generates all handshaking signals needed to implement compatibility mode of the parallel port communication protocol.

When this mode is enabled, the PPIC automatically generates a BUSY signal to receive the leading edge of nSTROBE from the host, and latches the logic levels on PPD7-PPD0 pins into the PPDATA register. The PPIC then waits for nSTROBE to negate it and for the PPDATA's data field to be read. After the PPDATA is read, the PPIC asserts nACK for the duration specified in the ACK Width Register (PPACKWTH), and then negates the nACK and BUSY signal to conclude the data transfer, as shown in Figure 10-1.

NOTE

The BUSY-control bit's initial value in the PPSTAT register, PPSTAT[3], which is "1" after a system reset, results in the high logic level on BUSY output and handshaking disable. To enable hardware handshaking in this mode, the BUSY-control bit PPSTAT[3] must be cleared to "0" by software beforehand.

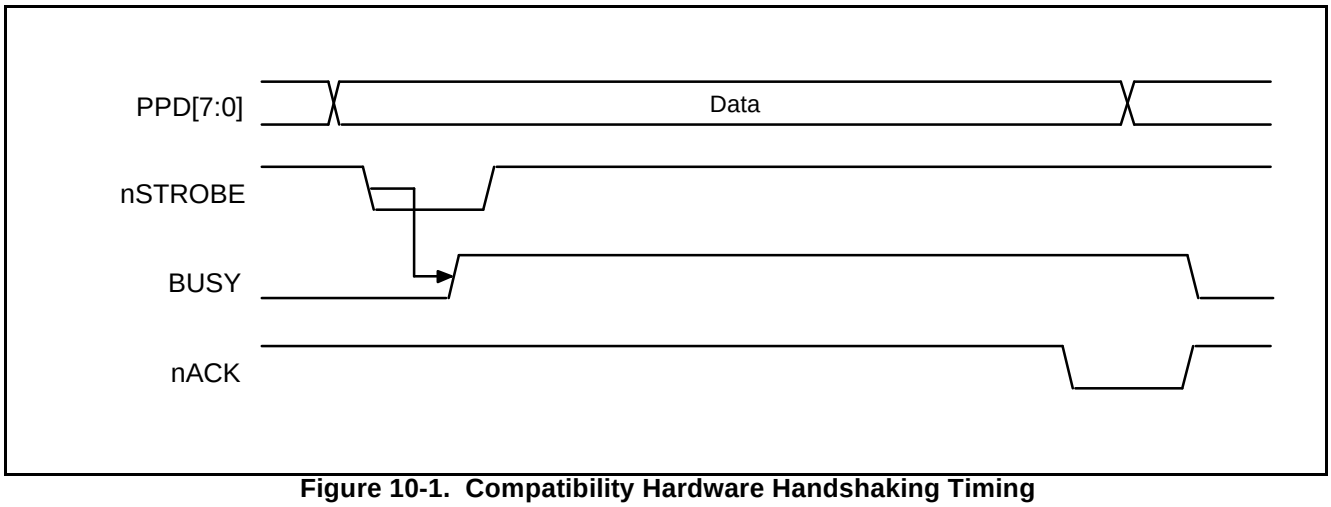


Figure 10-1. Compatibility Hardware Handshaking Timing

ECP-WITHOUT-RLE MODE

ECP-without-RLE hardware handshaking mode is enabled by setting the PPCON's mode-selection bits to "10", i.e. PPCON[3:2] = 10. In this mode, hardware generates handshaking signals needed to implement ECP mode of the parallel port communication protocol.

When receiving data from the host, the PPIC automatically responds to the high-to-low transition on the nSTROBE by latching the logic levels on the PPD7-PPD0 and nAUTOFD in the PPDATA register. The nAUTOFD logic level, which is latched to the PPDATA[8], indicates whether the current data on the PPD[7:0] is a data-byte or a command-byte. When the PPDATA is read, the PPIC drives BUSY to high level and waits for nSTROBE to go high level. It then drives BUSY to low level to conclude one forward data transfer operation, as shown in Figure 10-2.

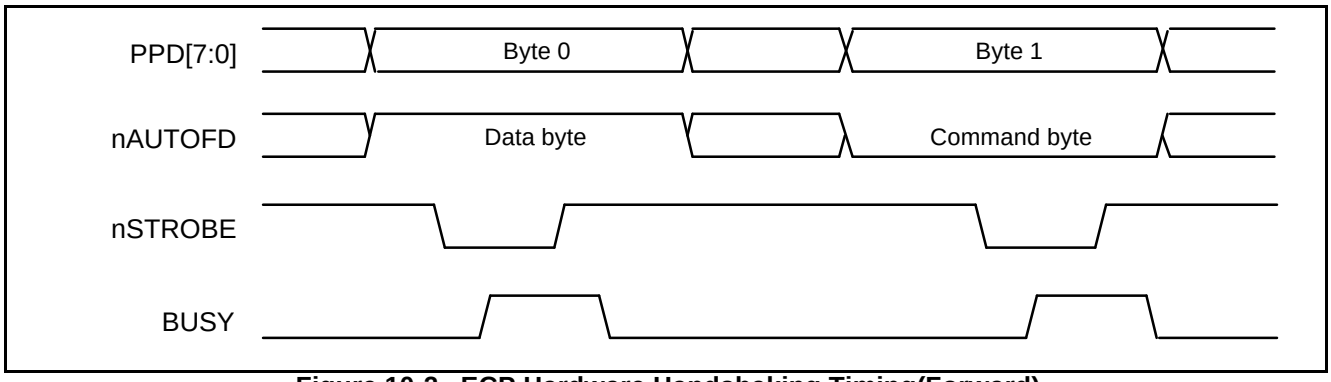


Figure 10-2. ECP Hardware Handshaking Timing(Forward)

The reception of a command byte, indicated by PPDATA[8]=0, causes the command received-bit in the PPIC interrupt pending register, PPINTPND[9], to be set to "1". By examining the PPDATA[7], software will interpret the command byte as a channel address if it is "1" and carry out the corresponding operation, or interpret the command byte as a run-length count if it is "0" and then perform data decompression.

During reverse data transfers, software is responsible for data compression, and writing data or command byte in PPDATA to define the logic levels on PPD7-PPD0 and BUSY pins. The PPDATA[8] indicates whether the current data on the PPDATA[7:0] is a data-byte or a command-byte. The state of PPDATA[8] is output through the BUSY pin. In response to writing the PPDATA, the PPIC automatically drives the nACK to low level and waits for the nAUTOFD to go to high level. It then drives nACK to high level to conclude one reverse data transfer operation, as shown in Figure 10-3.

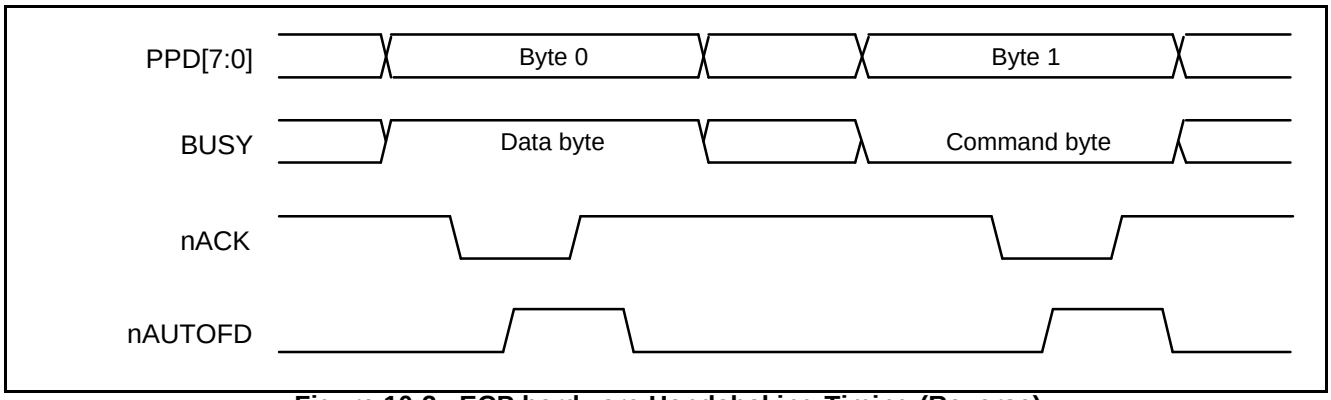


Figure 10-3. ECP hardware Handshaking Timing (Reverse)

ECP-WITH-RLE MODE

ECP-with-RLE hardware handshaking mode is enabled by setting the PPCON's mode-selection bits, PPCON[3:2], to “11”. In this mode, the PPIC performs the same ECP mode handshaking as in ECP-without-RLE mode, except for the fact that run-length compression/decompression is also carried out by hardware.

During forward data transfers, the PPIC automatically detects and intercepts run-length counts, and carries out data decompression. Only the channel addresses will cause the command-received-bit in the PPINTPND register, PPINTPND[9], to be set to one. If the command-receive interrupt occurs in ECP-with-RLE mode, the software performs the operations associated with the channel address.

Similarly, the PPIC automatically carries out the data compression in PPDATA during the reverse data transfers.

DIGITAL FILTERING

The KS32C6400 provides digital filtering function on host control signal inputs, nSELECTIN, nSTROBE, nAUTOFD and nINIT, to improve noise immunity and make the PPIC more impervious to the inductive switching noise. The digital filtering function can be enabled regardless of hardware handshaking or software handshaking.

If this function is enabled, the host control signal can be detected only when its input level keeps stable during two sampling periods.

Digital filtering can be disabled to avoid signal missing in some specialized applications with high bandwidth requirement. Otherwise, it is recommended that digital filtering be enabled.

PPIC SPECIAL REGISTERS

PARALLEL PORT DATA REGISTER

The parallel port data register, PPDATA, contains an 8-bit data field, PPDATA[7:0], that defines the logic level on the parallel port data pins, PPD[7:0]. It also contains a status bit, PPDATA[8], which is used to indicate when a command byte (RLE count or channel address) is received during forward data transfers in ECP mode.

Register	Offset Address	R/W	Description	Reset Value
PPDATA	0xB000	R/W	Parallel port data register	0x00000100

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									x	DATA FIELD							

[7:0] Data for parallel port bus, PPD[7:0]

This is an 8bit read/write field. When PPCON[6] is zero and this field is read, this field provides the logic levels on the PPN[7:0], which is latched when the strobe input from the host(nSTORBE) transits from high to low level. (The PPCON[6] bit determines the forward or reverse dataflow direction of the parallel port.) When PPCON[6] is one and this field is written, the value of this field determines the logic level on the PPD[7:0].

[8] ECP mode command byte indicator

During the ECP forward data transfers, the logic level of the nAUTOFD is read from PPDATA[8]. The nAUTOFD indicates whether the data in the PPDATA[7:0] is a data-byte or a command-byte. To read the nAUTOFD from the PPDATA[8], the following two conditions are required:

1. nSTROBE has transited from high level to low level.
2. The data bus output enable bit in the parallel port control register, PPCON[6], is 0.

When the ECP data transfers are in reverse and the data bus output enable bit in the parallel port control register, PPCON[6], is 1, the logic level of the BUSY pin is written to PPDATA[8]. The BUSY pin indicates whether the data written in the PPDATA[7:0] is a data-byte or a command-byte.

0 = Command-byte in the PPDATA[7:0]
1 = Data-byte in the PPDATA[7:0]

PARALLEL PORT STATUS REGISTER

The parallel port status register, PPSTAT, contains eleven bits to control the parallel port interface signals. These eleven bits consist of four read-only bits to read the logic level of the host input pins, two read-only bits to read the logic level on the BUSY and nACK output pins, and five read/write bits to control the logic levels on the printer output pins by software for handshaking control.

Register	Offset Address	R/W	Description	Reset Value
PPSTAT	0xB004	R/W	Parallel port status register	0x000007E8

31	11	10	9	8	7	6	5	4	3	2	1	0
		x	x	x	x	x	x	x	x	x	x	x

[0] nFAULTcontrol — Setting this bit drives the nFAULT output to low level; clearing it drives the signal high level on the external nFAULT pin. The nFAULT informs the host of a fault condition in the printer engine.

0 = nFAULT output High; no printer fault
1 = nFAULT output Low; printer fault occurred

[1] SELECT control — Setting this bit to one drives the SELECT output to High level; clearing it to zero drives the signal low on the external SELECT pin. The SELECT informs the host of a response from the printer engine

0 = SELECT output Low; no response from printer
1 = SELECT output High; response received from printer

[2] PERROR control — Setting this bit drives PERROR output to high level; clearing it drives the signal low level on the external PERROR pin. The PERROR informs the host that a paper error has occurred in the engine.

0 = PERROR output Low; no paper error
1 = PERROR output High; paper error has occurred

[3] BUSY control — Setting this bit drives the external BUSY output to high level by force. This disables hardware handshaking. When this bit is zero, the external BUSY output is the internal BUSY signal.

0 = BUSY output Low; not busy
1 = BUSY output High; busy

[4] nACK control — Setting this bit drives the external nACK output to high level by force. This is generally done when hardware handshaking is disabled. When this bit is one, the external nACK is the internal nACK signal.

0 = nACK output High; acknowledge handshake
1 = nACK output Low; do not acknowledge handshake

[5] BUSY status — This read-only bit reflects the logic level on the external BUSY output pin. After a system reset, the PPSTAT[3] is "1", which results in one, the value of PPSTAT[5] being "1". So, for compatibility mode operation, you must clear the PPSTAT[3] by software beforehand so as to enable the hardware handshaking.

[6] nACK status — This read-only bit reflects the inverted logic level on the external nACK output pin. After a system reset, PPSTAT[6] is "1".

[7] nSLCTIN status — This read-only bit reflects the level read on the nSLCTIN input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCON[1], is set to one.

[8] nSTROBE status — This read-only bit reflects the level read on the nSTROBE input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCON[1], is set to one.

[9] nAUTOFD status — This read-only bit reflects the level read on the nAUTOFD input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCON[1], is set to one.

[10] nINIT status — This read-only bit reflects the level read on the nINIT input pin after synchronization and optional digital filtering when the digital filtering enable bit, PPCON[1], is set to one.

PARALLEL PORT ACK WIDTH REGISTER

This register contains the 9-bit nACK pulse width field. This value defines the nACK pulse width whenever the parallel port interface controller enters Compatibility mode, that is, when the parallel port control register mode bits, PPCON[3:2], are set to “01”. The nACK pulse width is selectable from 0 to 511 MCLK periods.

The nACK pulse width can be modified at any time and with any PPIC operation mode selection, but it can only be used during a compatibility handshaking cycle. If you change the nACK width near the end of a data transfer (when nACK is already low), the new pulse width value does not affect the current cycle. The new pulse width value would be used at the start of the next cycle.

Register	Offset Address	R/W	Description	Reset Value
PPACKWTH	0xB008	R/W	Parallel port acknowledge width register	Undef.

31 … 9	8 … 0
	nACK PULSE WIDTH

[8:0] nACK pulse width — The value in this 9-bit field defines the nACK pulse width when Compatibility mode is enabled(PPCON[3:2] = '01') The period of the nACK pulse can range from 0 to 511 MCLKs. If you write a new value to the nACK width field near the end of a data transfer operation, the new pulse width value does not take effect until the next cycle takes place.

PARALLEL PORT CONTROL REGISTER

The parallel port control register, PPCON, is used to configure the PPI operations, such as handshaking, digital filtering, operating mode, data bus output, abort operations, and DMA. The PPCON[15:13] bits are read-only.

Register	Offset Address	R/W	Description	Reset Value
PPCON	0xB00C	R/W	Parallel port control register	0x00000000

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		x	x	x	x	-	x	x	x	x	x	x	x	MODE		x	x

[0] Software reset

Setting the software reset bit causes the PPIC's handshaking control and compression/decompression logic to immediately terminate the current operation and return to software Idle state. When PPCON[0] is set to "1", the run-length decompression status bit, PPCON[13], and the full status bit, PPCON[14], are automatically cleared to "0".

0 = No effect
1 = Terminate current PPIC operation and enter idle status

[1] Digital filter enable

Setting this bit enables digital filtering on all four host control signal inputs: nSELECTIN, nSTROBE, nAUTOFD, and nINIT.

0 = disable
1 = enable

[3:2] Mode selection

This two-bit value selects the current operating mode of the parallel port interface.

Software mode: disables all hardware handshaking so that handshaking can be performed by software.

Compatibility mode: Compatibility mode hardware handshaking can be enabled during a forward data transfer.

You can change the mode selection at any time, but if a Compatibility mode operation is currently in-progress, it will be completed as a normal operation. Mode should be changed from Compatibility mode to another mode only when BUSY is high level. This ensures that there is no parallel port activity while the parallel port is being re-configured.

ECP-without-RLE mode: ECP mode hardware handshaking without RLE support can be enabled during forward or reverse data transfers. You can change the mode selection at any time, but if an ECP cycle is currently in progress, it will be completed as a normal operation.

ECP-with-RLE mode: ECP mode hardware handshaking with RLE support can be enabled during forward or reverse data transfers. Changing the mode doesn't affect current data transfer operation, including compression/decompression, until data transfer operation is completed. To abort an operation immediately, set the software-reset bit, PPCON[0], to "1".

00 = Software mode
01 = Compatibility mode
10 = ECP mode without RLE
11 = ECP mode with RLE

[4] ECP direction

This bit determines whether the direction of ECP is forward or reverse. If this bit is set to '1', then the reverse direction is operated.

0 = forward
1 = reverse

[5] Error cycle

The error cycle bit is used to execute an error cycle in compatibility mode. When PPCON[5] is set to "1", the BUSY status bit in the parallel port status register, PPSTAT[5], is set to "1". This immediately causes the KS32C6400 to drive the BUSY to high level. If you set the error cycle bit while a compatibility mode handshaking sequence is in progress, the PPSTAT[5] will remain to be set to one beyond the end of the current cycle.
The error cycle bit does not affect the nACK pulse if it is already active, but it will delay an nACK pulse if it is about to be generated. When PPCON[5] is "1", software can set or clear the parallel port status register control bits: PPSTAT[0] (nFAULT control), PPSTAT[1] (SELECT control), and PPSTAT[2] (PERROR control). When PPCON[5] is cleared to “0", the parallel port interface controller generates a delayed nACK pulse and makes BUSY low active to finish the error cycle

0 = Generate nACK and negate BUSY; end error cycle
1 = Execute an error cycle(drive BUSY high level)

[6] Data bus output enable

The parallel port data bus output enable bit performs two functions:
1) It controls the state of the tri-state output drivers.
2) It qualifies the data latching from the output drivers into the parallel port data register’s data field, PPDATA[7:0].

When PPCON[6] is "0", the parallel port data bus lines, PPD[7:0] are disabled. This allows data to be latched onto the PPDATA’s data field. When PPCON [6] is "1", the PPD[7:0] is enabled and data is prevented from being latched onto the PPDATA’s data field. In this frozen state, the data field is unaffected by the transition of nSTROBE.

The setting of the abort bit, PPCON[7], affects the operation of the data bus output enable bit, PPCON[6]. If PPCON[7] is "1", the nSELECTIN must remain high to allow PPCON[6] to be set, or to remain set. If PPCON[6] is "1" and nSELECTIN goes low, the PPCON[6] is cleared and setting this bit will have no effect.

0 = Disable PPD[7:0] output
1 = Enable PPD[7:0] output

[7] Abort

The abort bit causes the parallel port interface controller to use nSELECTIN to detect the time when the host suddenly aborts a reverse transfer and returns to compatibility mode; If PPCON[7] is "1", the low level on nSELECTIN causes the parallel port data bus output enable bit PPCON[6] to be cleared, and the output drivers for the data bus lines PPD[7:0] to be tri-stated.

0 = Normal operation
1 = Disable data bus output and tri-state PPD[7:0] drivers

[8] DMA selection

The PPIC can issue a DMA request in compatibility mode, ECP-without-RLE mode, or in ECP-with-RLE mode, if the DMA request enable bit PPCON[9] is set to one. The DMA selection bit determines which DMA channel is used for data transfer. When PPCON[8] is "0", the DMA channel 1 is used; when it is "1", the DMA channel 2 is used.

0 = DMA1
1 = DMA2

[9] DMA request enable	When this bit is set to "1", the PPIC issues a DMA request to DMA channel 1 or 1 during a data transfer. Otherwise, an interrupt is requested for the data transfer. 0 = Generate interrupt request for data transfer 1 = Send DMA request to DMA1/DMA2 for data transfer
[10] Flush request	When this bit is set to "1", the PPIC issues a flush request to send the remaining data to the parallel port. The remaining data is the run-length code and the data in the PPIC's buffer while reverse ECP-with-RLE mode is operating. 0 = No operation 1 = Remained data to be transmitted to PPIC
[12] Zero insert	When the run-length count is '0', this bit specifies whether or not to send the RLE count during ECP-with-RLE reverse data transfers. If this bit is set to '1', then the count "0" will be sent. Otherwise, it will not be sent. 0 = When run-length is 0, only data to be transmitted 1 = When run-length is 0, nCMD and data are transmitted
[13] RLE status	This bit indicates that the run-length decompression is taking place during forward data transfers in ECP-with-RLE mode. It is set when a run-length count is received and loaded into the internal counter, and cleared when the last read of the PPDATA data field occurs. 0 = finished 1 = decompression is operating
[14] Data latch status	If a data is latched to PPDATA, then this bit is set to '1'. It is automatically cleared to zero when the PPDATA is read. 0 = No data 1 = Data is latched, if this bit is read, automatically cleared
[15] Data empty	In reverse ECP mode, this bit specifies the PPDATA is empty. It is automatically cleared to zero while the PPDATA is written with a new data. 0 = Data is processed 1 = PPDATA buffer is empty, it is automatically cleared when write operation is occurred

PARALLEL PORT INTERRUPT EVENT REGISTER

The two parallel port interrupt event registers, PPINTEN and PPINTPND, control interrupt-related events for the input signal originating from the host, as well as data reception, command reception, and invalid events. Enable register, PPINTEN, contains the interrupt enable bits for each interrupt event that is indicated by the PPINTPND status bits. If its PPINTEN enable bit is "1", the corresponding event causes the JupiterII CPU to generate an interrupt request. Otherwise, no interrupt request is issued.

Register	Offset Address	R/W	Description	Reset Value
PPINTEN	0xB010	R/W	Parallel port interrupt enable register	0x00000000

31	12	11	10	9	8	7	6	5	4	3	2	1	0
		x	x	x	x	x	x	x	x	x	x	x	x

[0] nSLCTIN Low-to-High — The bit of PPINTPND is set when a Low-to-High transition on nSLCTIN is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Disable interrupt
1 = Enable interrupt

[1] nSLCTIN High-to-Low — The bit of PPINTPND is set when a High-to-Low transition on nSLCTIN is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Disable interrupt
1 = Enable interrupt

[2] nSTROBE Low-to-High — The bit of PPINTPND is set when a Low-to-High transition on nSTROBE is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Disable interrupt
1 = Enable interrupt

[3] nSTROBE High-to-Low — The bit of PPINTPND is set when a High-to-Low transition on nSTROBE is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Disable interrupt
1 = Enable interrupt

[4] nAUTOFD Low-to-High — The bit of PPINTPND is set when a Low-to-High transition on nAUTOFD is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Disable interrupt
1 = Enable interrupt

[5] nAUTOFD High-to-Low — The bit of PPINTPND is set when a High-to-Low transition on nAUTOFD is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Disable interrupt
1 = Enable interrupt

[6] nINIT Low-to-High — The bit of PPINTPND is set when a Low-to-High transition on nINIT is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Disable interrupt
1 = Enable interrupt

[7] nINIT High-to-Low	The bit of PPINTPND is set when a High-to-Low transition on nINIT is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated. 0 = Disable interrupt 1 = Enable interrupt
[8] Data received	The bit of PPINTPND is set when data is latched into the PPDATA register*s data field. This occurs on every High-to-Low transition of nSTROBE when the parallel port data bus enable bit, PPCON[6]. is "0". An interrupt is also generated if ECP-with-RLE mode is enabled, and if a data decompression is in progress. 0 = Disable interrupt 1 = Enable interrupt
[9] Command received	The bit of PPINTPND is set when a command byte is latched into the PPDATA register data field. If ECP-without-RLE mode is enabled, a command received interrupt is issued whenever a run-length or channel address is received. If ECP-with-RLE mode is enabled, a command received interrupt is issued only when a channel address is received. This event can be posted only when ECP mode is enabled. The corresponding enable bit in the PPINTEN register determines whether an interrupt request will be generated when a command byte is received. 0 = Disable interrupt 1 = Enable interrupt
[10] Invalid transition	The bit of PPINTPND is set when nSLCTIN transitions High-to-Low in the middle of an ECP forward data transfer handshaking sequence. This interrupt is issued if nSLCTIN is Low when nSTROBE is Low or when BUSY is High. This event can only be detected when ECP mode is enabled. 0 = Disable interrupt 1 = Enable interrupt
[11] Transmit Data Empty	The bit of PPINTPND is set when the transmit data register(=PPDATA) can be written in the middle of an ECP reverse data transfer handshake sequence. 0 = Disable interrupt 1 = Enable interrupt

PARALLEL PORT INTERRUPT PENDING REGISTER

Register	Offset Address	R/W	Description	Reset Value
PPINTPND	0xB014	R/W	Parallel port interrupt pending register	0x00000000

31	12	11	10	9	8	7	6	5	4	3	2	1	0
		x	x	x	x	x	x	x	x	x	x	x	x

[0] nSLCTIN Low-to-High — The bit of PPINTPND is set when a Low-to-High transition on nSLCTIN is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Normal operation
1 = Transition occured; issue interrupt if enabled in PPINTEN

[1] nSLCTIN High-to-Low — The bit of PPINTPND is set when a High-to-Low transition on nSLCTIN is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Normal operation
1 = Transition occured; issue interrupt if enabled in PPINTEN

[2] nSTROBE Low-to-High — The bit of PPINTPND is set when a Low-to-High transition on nSTROBE is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Normal operation
1 = Transition occured; issue interrupt if enabled in PPINTEN

[3] nSTROBE High-to-Low — The bit of PPINTPND is set when a High-to-Low transition on nSTROBE is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Normal operation
1 = Transition occured; issue interrupt if enabled in PPINTEN

[4] nAUTOFD Low-to-High — The bit of PPINTPND is set when a Low-to-High transition on nAUTOFD is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Normal operation
1 = Transition occured; issue interrupt if enabled in PPINTEN

[5] nAUTOFD High-to-Low — The bit of PPINTPND is set when a High-to-Low transition on nAUTOFD is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Normal operation
1 = Transition occured; issue interrupt if enabled in PPINTEN

[6] nINIT Low-to-High — The bit of PPINTPND is set when a Low-to-High transition on nINIT is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated.

0 = Normal operation
1 = Transition occured; issue interrupt if enabled in PPINTEN

[7] nINIT High-to-Low	The bit of PPINTPND is set when a High-to-Low transition on nINIT is detected. If the corresponding enable bit is set in the PPINTEN register, an interrupt request is generated. 0 = Normal operation 1 = Transition occured; issue interrupt if enabled in PPINTEN
[8] Data received	The bit of PPINTPND is set when data is latched into the PPDATA register*s data field. This occurs on every High-to-Low transition of nSTROBE when the parallel port data bus enable bit, PPCON[6]. is "0". An interrupt is also generated if ECP-with-RLE mode is enabled, and if a data decompression is in progress. 0 = Normal operation 1 = Data received; issue interrupt if enabled in PPINTEN
[9] Command received	The bit of PPINTPND is set when a command byte is latched into the PPDATA register data field. If ECP-without-RLE mode is enabled, a command received interrupt is issued whenever a run-length or channel address is received. If ECP-with-RLE mode is enabled, a command received interrupt is issued only when a channel address is received. This event can be posted only when ECP mode is enabled. The corresponding enable bit in the PPINTEN register determines whether an interrupt request will be generated when a command byte is received. 0 = Normal operation 1 = Command byte received; issue interrupt if enabled in PPINTEN
[10] Invalid transition	The bit of PPINTPND is set when nSLCTIN transitions High-to-Low in the middle of an ECP forward data transfer handshaking sequence. This interrupt is issued if nSLCTIN is Low when nSTROBE is Low or when BUSY is High. This event can only be detected when ECP mode is enabled. 0 = Normal operation 1 = Transition occured; issue interrupt if enabled in PPINTEN
[11] Transmit Data Empty	The bit of PPINTPND is set when the transmit data register(=PPDATA) can be written in the middle of an ECP reverse data transfer handshake sequence. 0 = Normal operation 1 = PPDATA empty; issue interrupt if enabled in PPINTEN

11 UART

INTRODUCTION

The KS32C6400 UART (Universal Asynchronous Receiver and Transmitter) unit provides two independent asynchronous serial I/O (SIO) ports, each of which can operate in interrupt-based or DMA-based mode, i.e. SIO can generate interrupt or DMA request to transfer data between CPU and SIO.

The KS32C6400 UART includes programmable baud-rates, infra-red (IR) transmit/receive, one or two stop bit insertion, 5-bit, 6-bit, 7-bit or 8-bit data transfers, and parity checking.

Each SIO contains a baud-rate generator, transmitter, receiver and control unit, as shown in Figure 11-1. The baud-rate generator can be clocked by either the internal system clock (MCLK). The transmitter and the receiver contain data buffer registers and data shifters. Data, which is to be transmitted, is written to the transmit holding register and then copied to the transmit shifter. It is then shifted out by the transmit data pin (TXDn). The received data is shifted by the receive data pin (RXDn), and then copied to the receive buffer register from the shifter once one data byte has been received. The control unit provides controls for mode selection and status/interrupt generation.

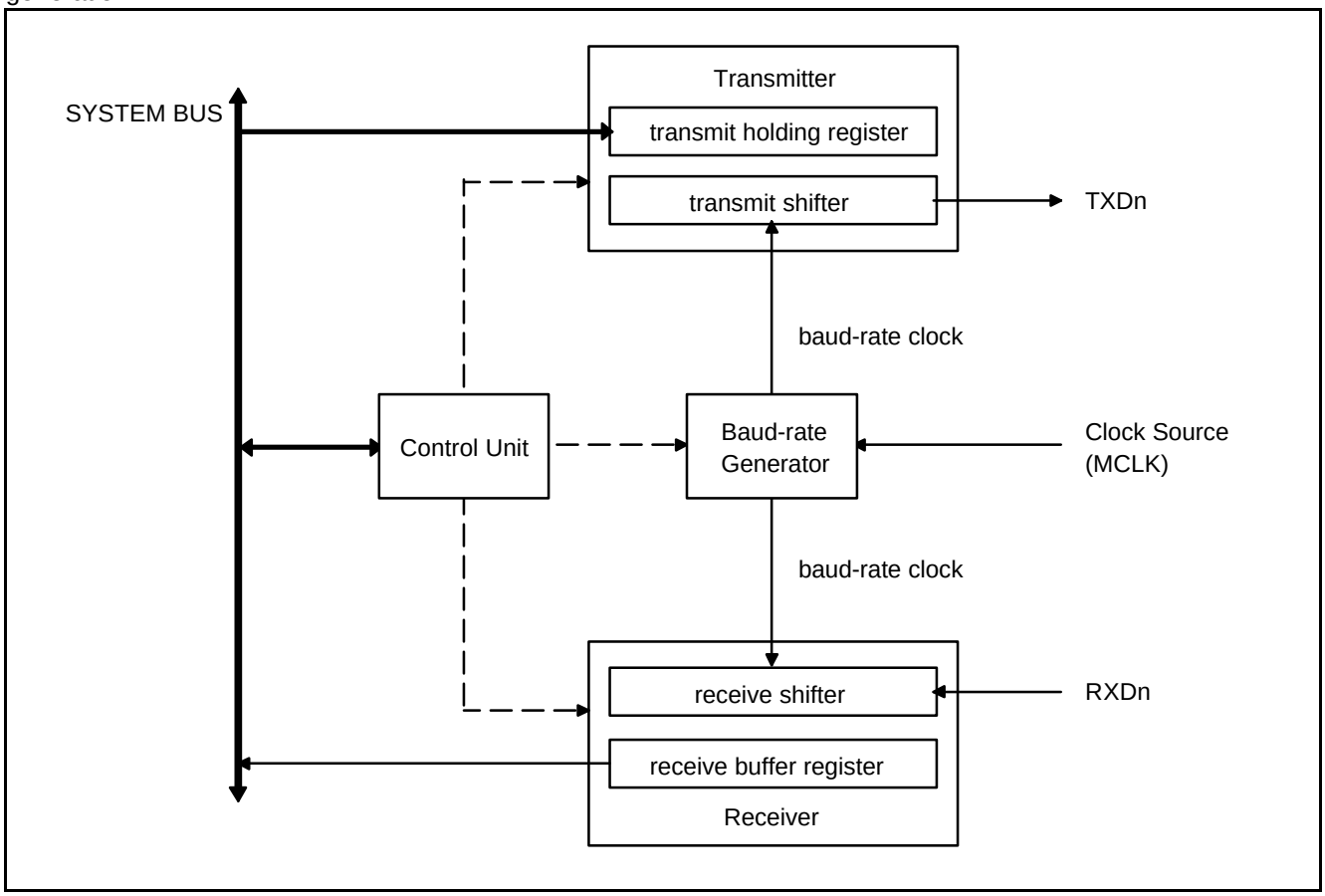


Figure 11-1. Serial I/O Block Diagram

UART OPERATION

The following sections describe the UART operations that include infra-red mode, loopback mode, interrupt generation, baud-rate generation, data transmission, data reception and so on.

INFRA-RED MODE

The KS32C6400 UART block supports infra-red (IR) transmit and receive, which can be selected by setting the infra-red-mode bit in the line control register(ULCONn). The implementation of the mode is shown in Figure 11-2.

In IR mode, the transmit period is pulsed at a rate of 3/16 as is at the normal serial transmit rate (when the transmit data value in the UTXBUF register is zero); in IR receive mode, the receiver must detect the 3/16 pulsed period to recognize a zero value in the receive buffer register, URXBUF, as the IR receive data. (refer to the frame timing diagrams shown in Figure 11-7 and 11-8)

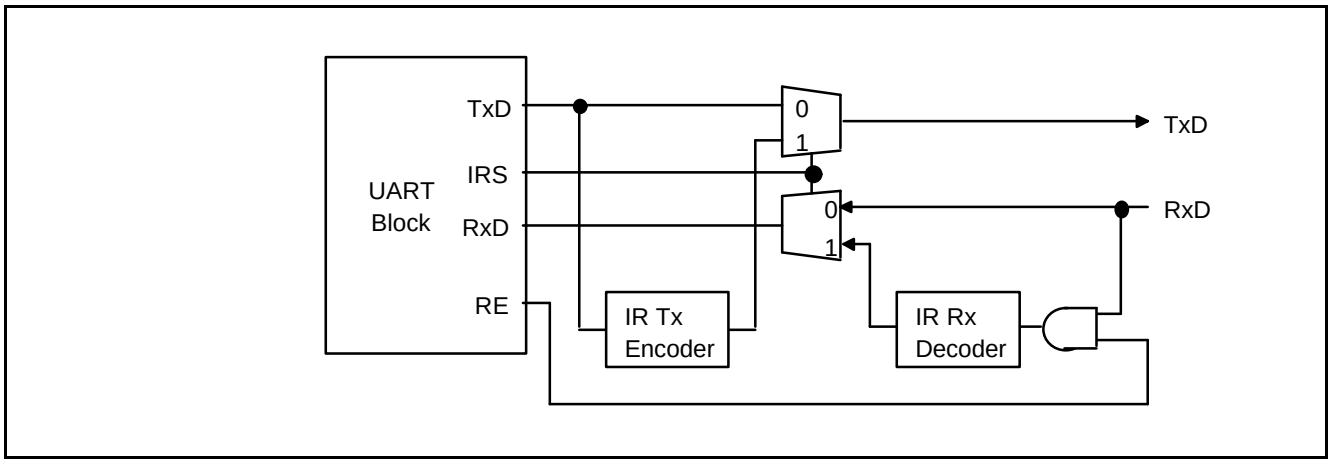


Figure 11-2. UART Block Diagram

LOOPBACK MODE

The KS32C6400 UART provides a test mode referred as the loopback mode to aid in isolating faults in the communication link. In this mode, the transmitted data is immediately received. This feature allows the processor to verify the internal transmit and to receive the data path of each SIO channel. This mode can be selected by setting the loopback-bit in the UART control register (UCONn).

INTERRUPT/DMA REQUEST GENERATION

Each SIO of the KS32C6400 UART has seven status signals: overrun error, parity error, frame error, break, receive buffer full, transmit buffer register empty and transmitter empty, all of which are indicated by the corresponding UART status register (USTATn).

The overrun error, parity error, frame error and break condition are referred to as the receive status, each of which can cause the receive status interrupt request, if the receive-status-interrupt-enable bit is set to one in the control register UCONn. When a receive-status-interrupt-request is detected, you can know the signal which causes the request by reading the status register (USTATn).

When the receiver transfers the data of the receive shifter to the receive buffer register, it activates the receive buffer full status signal which will cause the receive interrupt, if the receive mode in control register is selected as the interrupt mode. When the transmitter transfers data from its transmit holding register to its shifter, the transmit holding register empty status signal is activated. The signal causes the transmit interrupt if the transmit mode in control register is selected as interrupt mode.

The receive-buffer-full and transmit holding register empty status signals can also be connected to generate the DMA request signals if the receive/transmit mode in the control register is selected as the DMA mode.

As mentioned before, three DMA channels, DMA0, DMA1 and DMA2, are provided in the KS32C6400. Each SIO can be connected with a fixed DMA channel. In other words, the SIO0 can only generate the DMA0 request and the SIO1 can generate the DMA1 request or DMA2 request(CDMA).

BAUD-RATE GENERATION

Each SIO's baud-rate generator provides the serial clock for transmitter and receiver. The source clock for the baud-rate generator can be selected with the KS32C6400's internal system clock (MCLK), which is determined by the serial-clock-selection bit in UART line control register (ULCONn). The baud-rate clock is generated by dividing the source clock by 16 and a 16-bit divisor specified by the UART baud- rate divisor register (UBRDIVn). The UBRDIVn can be determined as follows:

UBRDIVn = (int)(source_clock / (bps x 16)) - 1

where the divisor should be from 1 to (2^{16}-1). For example, if the baud-rate is 56,000 bps and MCLK is 33 MHz (use internal system clock), UBRDIVn is:

UBRDIVn = (int)(33000000 / (56000 x 16)) - 1
= (int)(36.83) - 1
= 36 - 1 = 35

DATA TRANSMISSION

The data frame for transmission is programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits, which can be specified by the line control register (ULCONn). The transmitter can also produce the break condition. The break condition forces the serial output to logic 0 state for a duration longer than one frame transmission time. At the receiving end, the break condition sets an error flag as mentioned above.

DATA RECEPTION

Like the transmission, the data frame for reception is also programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits by settings in the line control register (ULCONn). The receiver can detect overrun error, parity error, frame error and break condition, each of which can set an error flag.

The overrun error indicates that new data has overwritten the old data before the old data has been read. The parity error indicates that the receiver has detected a parity condition other than what it was programmed for. The frame error indicates that the received data does not have a valid stop bit. The break condition indicates that the RXDn input is held in the logic 0 state for a duration longer than one frame transmission time.

UART LINE CONTROL REGISTER

There are two identical UART line control registers, ULCON0, 1, in the UART block, each for a SIO channel.

Register	Offset Address	R/W	Description	Reset Value
ULCON0	0xE000	R/W	UART channel 0 line control register	0x00000000
ULCON1	0xE800	R/W	UART channel 1 line control register	0x00000000

31	8	7	6	5	4	3	2	1	0
		IR		PARITY			ST	WL	

[1:0] Word Length (WL)

The word length indicates the number of data bits to be transmitted or received per frame.

00 = 5 bits
01 = 6 bits
10 = 7 bits
11 = 8 bits

[2] Number of stop bit

The number of stop bit specifies how many stop bits are used to signal end-of-frame (EOF).

0 = One stop bit per frame
1 = Two stop bit per frame

[5:3] Parity Mode (PMD)

The parity mode specifies how parity generation and checking are to be performed during UART transmit and receive operations.

0xx = No parity
100 = Odd parity
101 = Even parity
110 = Parity forced/checked as "1"
111 = Parity forced/checked as "0"

[7] Infra-Red Mode

The infra-red mode determines whether or not to use the infra-red mode.

0 = Normal Mode Operation
1 = Infra-red Tx/Rx Mode

NOTE: The ULCONn has to be configured before the UCONn is configured.

UART CONTROL REGISTER

There are two identical UART control registers (UCON0,1) in the UART block, each for a SIO channel. The UCONn has to be configured after the ULCONn is configured.

Register	Offset Address	R/W	Description	Reset Value
UCON0	0xE004	R/W	UART channel 0 control register	0x00000000
UCON1	0xE804	R/W	UART channel 1 control register	0x00000000

31	8	7	6	5	4	3	2	1	0
		LP	SB	-	TxM		RS	RxM	

[1:0] Receive Mode (RxM)

The RxM determines which function is currently able to read data from the UART receive buffer register, URXBUF. The difference between UCON0 and UCON1 should be noted. SIO0 can only generate a DMA0 request and SIO1 can only generate a DMA1 request or DMA2 request.

For UCON0	For UCON1
00 = disable SIO0	00 = disable SIO1
01 = interrupt request	01 = interrupt request
10 = DMA0 request	10 = DMA2 request
11 = not used	11 = DMA1 request

[2] Rx Status Interrupt Enable

This bit enables the UART to generate an interrupt if an exception, such as a break, frame error, parity error, or overrun error, occurs during a receive operation.
0 = do not generate receive status interrupt
1 = generate receive status interrupt

[4:3] Transmit Mode (TxM)

This TxM determines which function is currently able to write Tx data to the UART transmit holding register, UTXBUF. The difference between UCON0 and UCON1 should be noted. SIO0 can only generate a DMA0 request and SIO1 can only generate a DMA1 request or DMA2 request.

For UCON0	For UCON1
00 = disable SIO0	00 = disable SIO1
01 = interrupt request	01 = interrupt request
10 = DMA0 request	10 = DMA2 request
11 = not used	11 = DMA1 request

[6] Send Break

Setting UCON[6] causes the UART to send a break. The break is defined as a continuous low level signal on the transmit data output with a duration more than one frame transmission time.
This bit should be set to one when the transmitter is empty (transmitter empty bit, USTAT [7] ="1"). You can use the transmitter to measure a frame time interval. When the USTAT[7] is "1", write the transmit holding register, UTXBUF, with dummy data and then poll the USTAT[7] value. When it returns to "1", clear (reset) the send break bit UCON[6]. (You have sent the break for one frame time interval exactly.)
0 = not send break
1 = send break

[7] Loopback Bit

Setting loopback bit to 'one' causes the UART to enter loopback mode. In loopback mode, the TXDn pin is sent to the high level and the transmit holding register (UTXBUF) is internally connected to the receive buffer register (RBR). This mode is provided for test purposes only.
0 = Normal SIO operation mode
1 = Enable SIO loopback mode (only for testing)

NOTE: Even if you do not use the interrupt request, the interrupt request is selected to use SIO. If you don't need a interrupt request, you can disable the interrupt by configuring the interrupt mask register(INTMSK).

UART STATUS REGISTER

There are two identical UART control registers (UCON0,1) in the UART block, each for a SIO channel. The USTAT is read only register that is used to monitor the status of SIO.

Register	Offset Address	R/W	Description	Reset Value
USTAT0	0xE008	R/W	UART channel 0 status register	0x000000C0
USTAT1	0xE808	R/W	UART channel 1 status register	0x000000C0

31	8	7	6	5	4	3	2	1	0
		TE	TH	RD	-	BK	FR	PT	OV

[0] Overrun Error

USTAT[0] is automatically set to "1" whenever an overrun error occurs during a serial data receive operation. If the receive-status-interrupt- enable bit CON[2] is "1", and an overrun error occurs, a receive status interrupt will be generated. This bit is automatically cleared to "0" whenever the UART status register (USTAT) is read

0 = No overrun error during receive
1 = Overrun error(generate receive status interrupt if UCON[2] is "1")

[1] Parity Error

USTAT[1] is automatically set to "1" whenever a parity error occurs during a serial data receive operation. If the receive-status-interrupt-enable bit UCON[2] is "1", and a parity error occurs, a receive status interrupt will be generated. This bit is automatically cleared to "0" whenever the UART status register (USTAT) is read

0 = No parity error during receive
1 = Parity error(generate receive status interrupt if UCON[2] is "1")

[2] Frame Error

USTAT[2] is automatically set to "1" whenever a frame error occurs during a serial data receive operation. If the receive status-interrupt-enable bit UCON[2] is "1", and a frame error occurs, a receive status interrupt will be generated. The frame error bit is automatically cleared to "0" whenever the UART status register (USTAT) is read

0 = No frame error during receive
1 = Frame error(generate receive status interrupt if UCON[2] is "1")

[3] Break Interrupt

USTAT[3] is automatically set to "1" to indicate that a break signal has been received. If the receive status interrupt enable bit UCON[2] is "1", and a break occurs, a receive status interrupt will be generated. The break interrupt bit is automatically cleared to "0" when you read the UART status register.

0 = No break received
1 = Break received(generate receive status interrupt if UCON[2] is "1")

[5] Receive Data Ready

USTAT[5] is automatically set to "1" whenever the receive data buffer register (RBR) contains valid data received over the serial port. The receive data can then be read from the RBR. When this bit is "0", the RBR does not contain valid data. Depending on the current setting of the SIO receive mode bits, UCON[1:0], an interrupt or a DMA request is generated when USTAT[5] is "1".

0 = No valid data in the receive buffer register.
1 = Valid data present in the receive buffer register (issue interrupt or DMA request if SPCON[1:0] set)

[6] Tx Buffer Register Empty

USTAT[6] is automatically set to "1" when the transmit holding register (THR) does not contain valid data. In this case, the THR can be written with the data to be transmitted. When this bit is "0", the THR contains valid Tx data that has not been copied to the transmit shift register. In this case, the THR cannot be written with new Tx data. Depending on the current setting of the SIO transmit mode bits, UCON[4:3], an interrupt or a DMA request will be generated when USTAT[6] is "1".

0 = Valid data in transmit holding register
1 = No data in transmit holding register(issue interrupt or DMA request if UCON[1:0] set)

[7] Transmitter Empty

USTAT[7] is automatically set to "1" when the transmit holding register has no valid data to transmit and the Tx shift register is empty. When the transmitter empty bit is "1", it indicates that the transmitter function block is not used and you can manipulate the setting of the transmitter function block.

0 = Transmitter not empty; Tx in progress
1 = Transmitter empty; no data for Tx

UART TRANSMIT BUFFER REGISTER

There are two identical UART transmit holding registers, UTXBUF, in the UART block for two SIO channels, each of which contains an 8-bit data value to be transmitted over the SIO channel.

In DMA-based transmit mode, as the destination of the DMA channel, the address of the transmit holding register should be set to one, into the DMA destination address register.

Register	Offset Address	R/W	Description	Reset Value
UTXBUF0	0xE00C	W	UART channel 0 transmit holding register	Undef.
UTXBUF1	0xE80C	W	UART channel 1 transmit holding register	Undef.

31 ... 8	7	6	5	4	3	2	1	0
	TRANSMIT DATA							

[7:0] Transmit Data — This field contains the data to be transmitted by the corresponding SIO channel. When this register is written, the transmit holding register empty bit in the status register, USTAT[6], should be set to "0". This prevents overwriting transmit data that may already be present in the URXBUF. Whenever the UTXBUF is written with new value, the transmit register empty bit, USTAT[6], is automatically cleared to "0"

UART RECEIVE BUFFER REGISTER

There are two identical UART receive buffer registers, RBR, in the UART block for two SIO channels, each of which contains an 8-bit data value for received serial data.

In DMA-based receive mode, as the source of the DMA channel, the address of the receive buffer register should be set into the DMA source address register.

Register	Offset Address	R/W	Description	Reset Value
URXBUF0	0xE010	R	UART channel 0 receive buffer register	Undef.
URXBUF1	0xE810	R	UART channel 1 receive buffer register	Undef.

31 ... 8	7	6	5	4	3	2	1	0
	RECEIVE DATA							

[7:0] Receive Data — This field contains the data received from the corresponding SIO channel. When UART finishes receiving a data frame, the receive data ready bit in the UART status register, USTAT[5], should be set to "1". This prevents reading invalid receive data that may already be present in the URXBUF. Whenever the RBR is read, the receive data ready bit, USTAT[5], is automatically cleared to "0".

UART BAUD RATE DIVISOR REGISTER

The value stored in the baud rate divisor register, UBRDIV, is used to determine the serial Tx/Rx clock rate (baud rate) as follows:

UBRDIVn = (int)(source_clock / (bps x 16)) - 1

The source_clock is MCLK (the internal master clock).

Register	Offset Address	R/W	Description	Reset Value
UBRDIV0	0xE014	R/W	Baud rate divisor register 0	0x00000000
UBRDIV1	0xE814	R/W	Baud rate divisor register 1	0x00000000

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	BAUD RATE DIVISOR																

[15:0] Baud rate divisor — This field contains the baud rate divisor value for corresponding UART channel. Baud rate can be calculated as:

UBRDIVn = (int)(source_clock / (bps x 16)) -1

NOTE : The value of the baud-rate divisor should be from 1 to (2^{16}-1)

TIMING DIAGRAMS

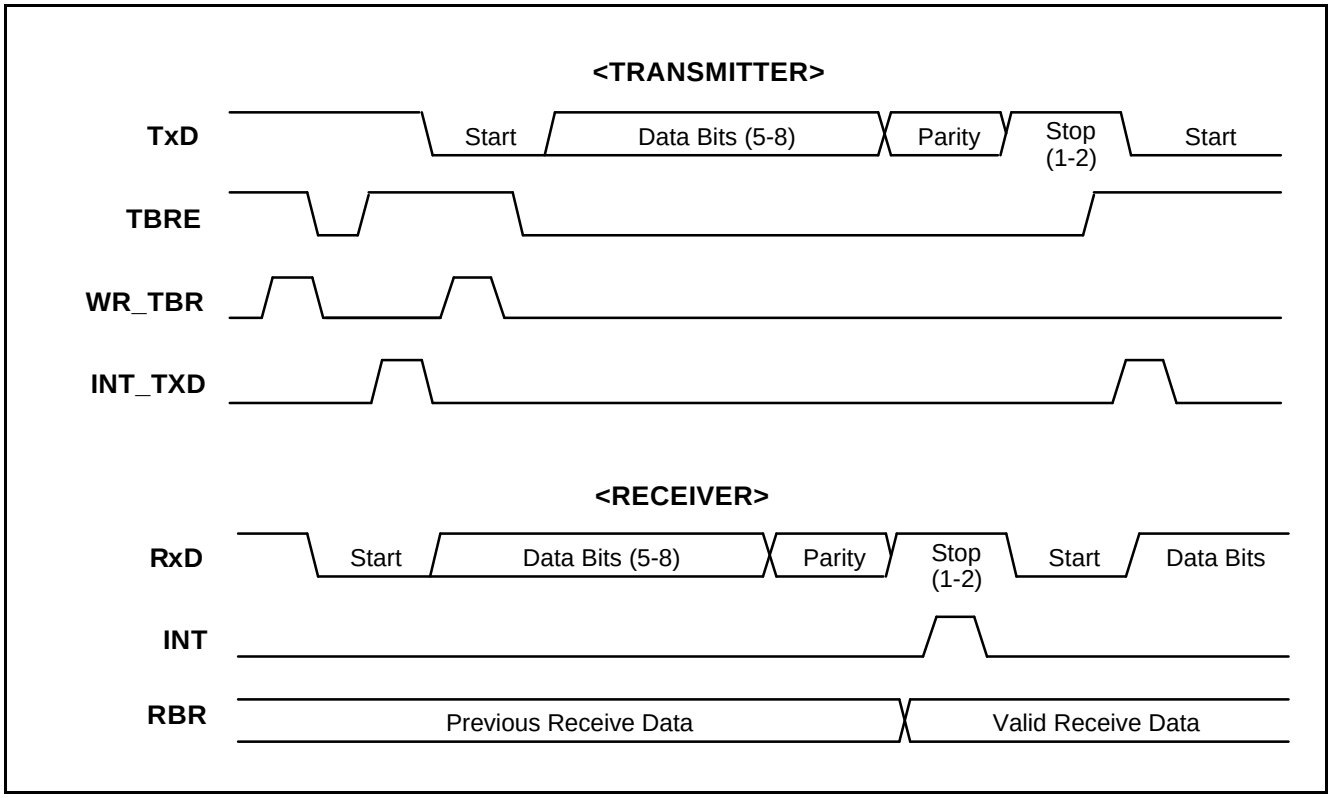


Figure 11-3. Interrupt-based Serial I/O Timing Diagram (Tx and Rx)

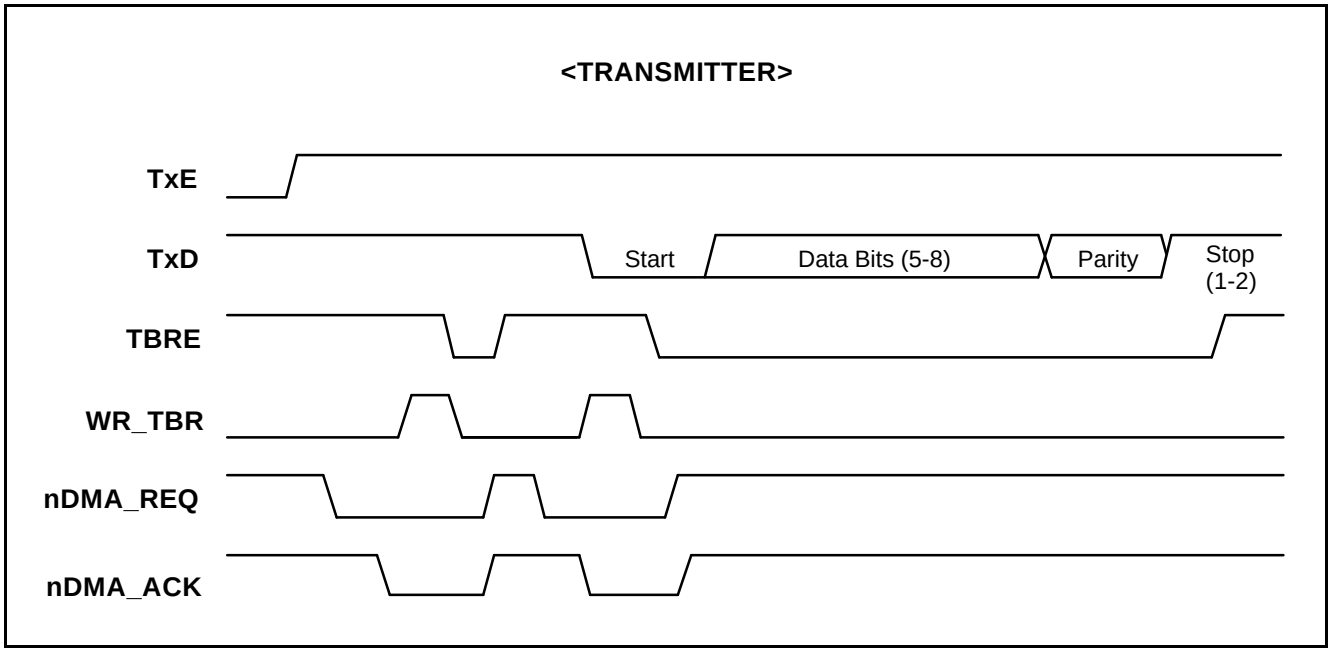


Figure 11-4. DMA-based Serial I/O Timing Diagram (Tx only)

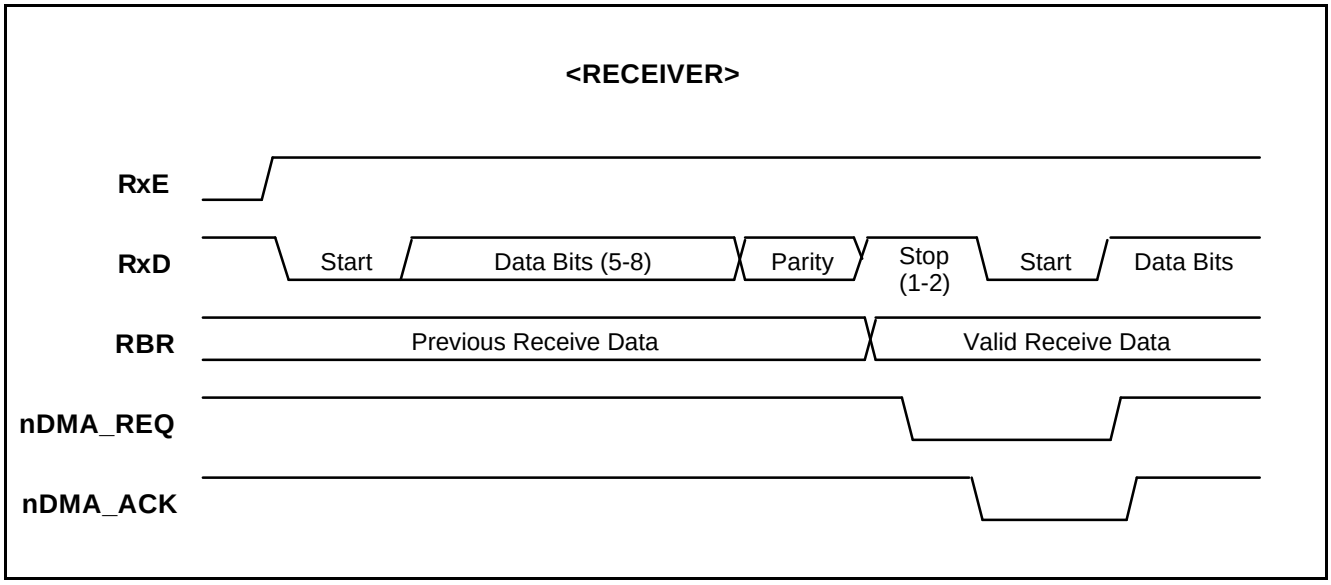


Figure 11-5. DMA-Based Serial I/O Timing Diagram (Rx only)

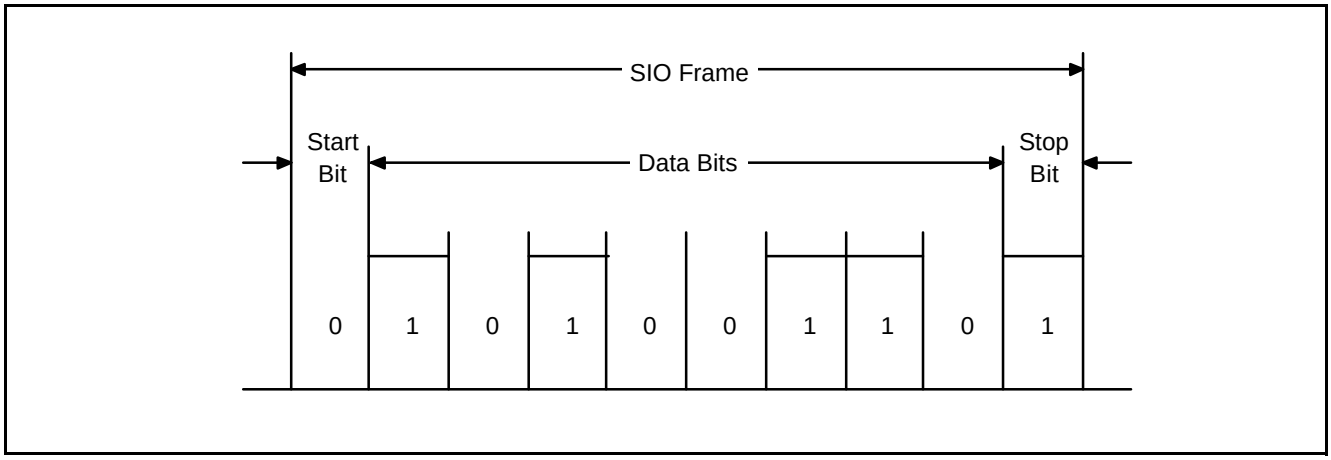


Figure 11-6. Serial I/O Frame Timing Diagram (Normal UART)

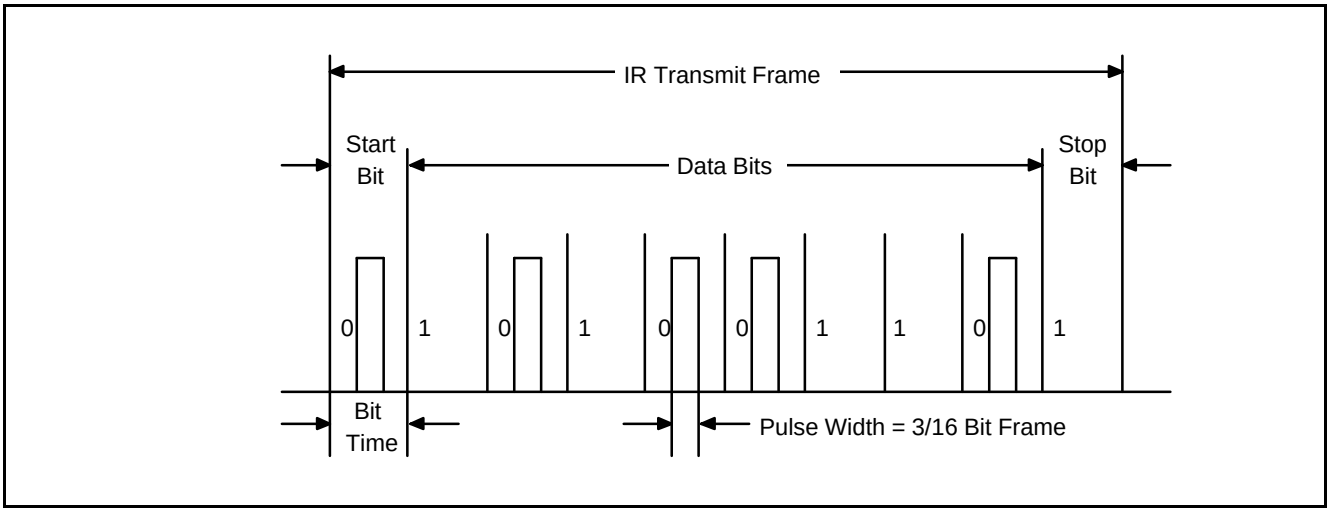


Figure 11-7. Infra-Red Transmit Mode Frame Timing Diagram

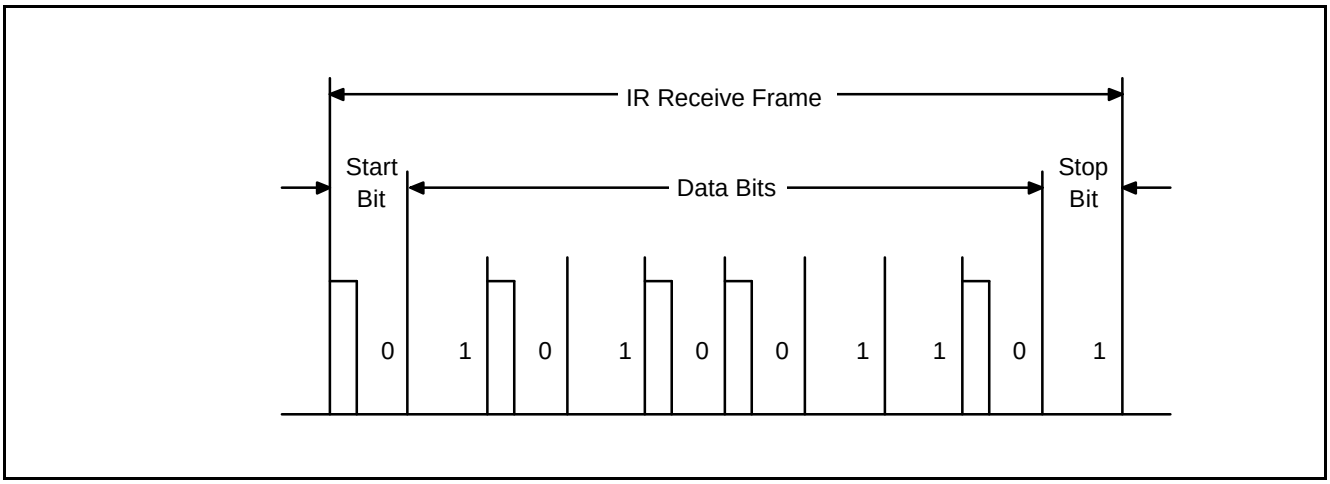


Figure 11-8. Infra-Red Receive Mode Frame Timing Diagram

12 TONE GENERATOR

INTRODUCTION

The KS32C6400 Tone Generator provides a programmable tone signal which has a 50% duty cycle. The tone signal can be used to make out 'keyclick' sound. The Tone Generator block has a tone counter which includes an 8-bit programmable divider and a 1/2 divider to make out the 50% duty cycle, and a Tone Data register (TONDATA) which has a tone enable/disable bit and tone count data bits. The 8-bit programmable divider receives MCLK/(prescaler+1)/128 clock signals and divides it depending on the count value of TONDATA [7:0] bits. You can set the prescaler value of TSTCON.

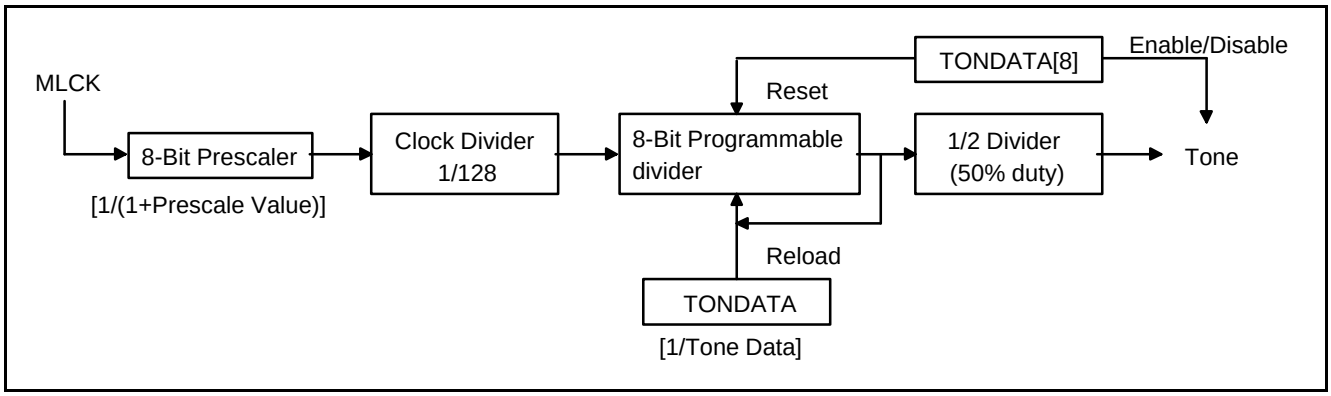


Figure 12-1. Tone Generator Block Diagram

TONDATA[8] bit enables/disables the Tone generator operation. When it is cleared to '0', the tone output is disabled (stopped) and the programmable divider is automatically cleared while the tone data register (TONDATA) retains its value. The initial value of the tone enable bit is '0'.

The input clock to the tone generator is MCLK/(prescaler+1)/128. The divided-by ratio of the tone counter is determined by the tone data register value, ranging from 0 to 255.

You have to load data into the tone data register (TONDATA) before enabling the tone generator to get the correct tone signal. To make out the 50% duty cycle tone signal, the KS32C6400 Tone Generator has a 1/2 divider with a programmable divider. The output of the programmable divider is divided by the 1/2 divider.

The frequency of the tone is calculated as follows:

$$\frac{\text{MCLK}}{(\text{Prescaler}+1) \times 128 \times \text{ToneData} \times 2}$$

Table 12-1. Tone Generator Data Value Setting (MCLK=33 MHz)

TONDATA	Tone Frequency	TONDATA	Tone Frequency
0	No tone (all high)	4	2.470 kHz
1	9.916 kHz	....	....
2	4.958 kHz	100	99 Hz
3	3.305 kHz	255	39 Hz

NOTE: The value of prescaler is 0xC.

TONE GENERATOR DATA REGISTER (TONDATA)

The tone generator data register (TONDATA) stores an 8-bit value which determines the frequency of the tone generator output. The value in the TONDATA register determines the divided-by ratio of the programmable divider. The divided-by value, therefore, ranges from 0 to 255. The output value of the tone counter is divided by two, producing a 50% duty tone output signal. A reset clears the TONDATA value to '00h'. The tone frequency is therefore calculated, based on the tone data value, as follows:

$$\frac{\text{MCLK}}{(\text{Prescaler}+1) \times 128 \times \text{ToneData} \times 2}$$

Register	Offset Address	R/W	Description	Reset Value
TONDATA	0xF004	R/W	Tone generator data register	0x000000FF

31	9	8	7	6	5	4	3	2	1	0
		x	TONE COUNT DATA							

[7:0] Tone Counter Data — 8-bit tone counter data value

[8] Tone generator control — 0 = Clear counters and reset tone output
1 = Generate tone

13 WATCH-DOG TIMER

INTRODUCTION

The KS32C6400 Watch-Dog Timer is used to resume controller operation when it is disturbed by malfunctions such as noise and system errors. It can be used as a normal interval timer to request interrupt services. You can set the prescaler value (initial value: 0xC) in TSTCON.

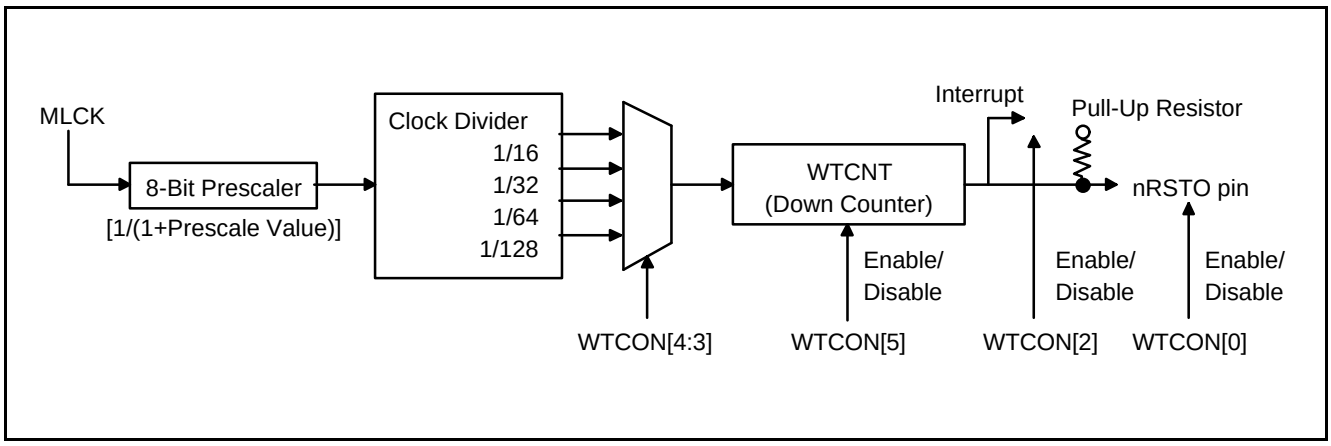


Figure 13-1. Watch-Dog Timer Block Diagram

WATCH-DOG TIMER CONTROL REGISTER

Using the Watch-Dog Timer Control register, WTCON, you can enable/disable the watch-dog timer, select the clock signal from 4 different sources, enable/disable interrupts, and enable/disable the watch-dog timer reset output signal through the nRSTO pin. If the counter value of the watch-dog timer is “0,” the WTCON is cleared to 0x00.

Register	Offset Address	R/W	Description	Reset Value
WTCON	0xF800	R/W	Watch-dog timer control register	0x00000021

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
												x	CLK		x	-	x

[0] Reset mode
0 = Disable nRSTO pin
1 = Enable nRSTO pin

[2] Interrupt mode
0 = Disable interrupt
1 = Enable interrupt

[4:3] Clock division factor selection
00 = 16
01 = 32
10 = 64
11 = 128

[5] Watch-dog timer enable/disable
0 = Disable timer
1 = Enable timer

WATCH-DOG TIMER COUNTER REGISTERS

The watch-dog timer counter register, WTCNT, is used to specify the time out duration. The watch-dog timer enable bit (bit 5, WTCON) must be '0' before loading a value to this register.

Watch-dog timer clock = MCLK / (prescale value + 1) / division factor
Watch-dog timer duration = count_value x watch-dog timer clock period

Register	Offset Address	R/W	Description	Reset Value
WTCNT	0xF804	R/W	Watch-dog timer count register	0x00000003

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		COUNT VALUE															

[15:0] Watch-dog timer counter This specifies the time out duration.

Table 13-1. Watch-Dog Timer Counter Setting (MCLK=33 MHz)

Clock Source	Resolution	Maximum Interval	Remark
MCLK/(prescale+1)/16	6.30 μs	413 ms	Default setting
MCLK/(prescale+1)/32	12.6 μs	826 ms	–
MCLK/(prescale+1)/64	25.2 μs	1.651 s	–
MCLK/(prescale+1)/128	50.4 μs	3.305 s	–

NOTE:The value of prescaler is 0xc and WTCNT is 16-bit count.

WATCH-DOG TIMER OPERATION

Before loading a count value into the Watch-Dog Timer Count Register, WTCNT, you have to disable the watch-dog timer by setting the WTCON[5] bit to zero. When WTCON[5] bit set to "1," the watch- dog timer is enabled and the counter starts down-count. The value of the watch-dog counter register is accessible at any time while the watch-dog timer is enabled, because it provides read and write features.

The watch-dog timer provides general timer interrupt as well as system reset features. To enable the watch-dog timer interrupt, the WTCON[2] bit has to be set to “1”. When the watch-dog timer interrupt is enabled, the interrupt signal generates one pulse of request signal to CPU. The interrupt pending bit (bit2, INTPNDR) is automatically set to '1' when an underflow occurs.

When WTCON[0] bit is '1', the nRSTO pin is enabled and watch-dog reset signal comes through the nRSTO pin. If watch-dog counter reaches to zero, for some reason, the nRSTO signal is activated during 128 MCLK cycles, and the WTCON will be automatically set to 0x0. To avoid watch-dog timer activating the nRSTO signal, the MCU has to reload the counter value into the watch-dog counter register (WTCNT) periodically.

The nRSTO signal is not connected to the nRESET internally. If nRSTO is connected to the nRESET by an external logic, the KS32C6400 initialization routine will be executed by the nRSTO signal.

NOTE

The pin type of nRST is open-drain output. If you want to use the nRST pin, a pull-up resistor must be installed on the nRST pins.

BLANK PAGE

14 I/O PORTS

INTRODUCTION

The KS32C6400 has 7 input, 13 output, 8 input/output, 10 extra-output ports and serial EERAM control ports. Some ports pins can be multiplexed with other internal device units, such as JTAG, UART, interrupt controller and so on.

I/O PORT SPECIAL REGISTERS

Two registers, IOPMOD and IOP, control the I/O port configuration. Table 14-1 shows the possible values for the port mode registers. The IOP register contains one bit for each port which reflects the signal level at the respective port pin.

Table 14-1. I/O Port Mode Configuration Settings

I/O Port Pin	I/O Port Mode Configuration Settings	
	Function for one	Function for zero
GIP0 / RXD0	GIP0	RXD0
GIP1 / RXD1	GIP1	RXD1
GIP2 / nEINT0	GIP2	nEINT0
GIP3 / nEINT1	GIP3	nEINT1
GIP4 / nEDREQ0	GIP4	nEDREQ0
GIP5 / nEDREQ1	GIP5	nEDREQ1
GIP6 / nEINT2	GIP6	nEINT2
GOP0 / TXD0	GOP0	TXD0
GOP1 / TXD1	GOP1	TXD1
GOP2 / nEDACK0	GOP2	nEDACK0
GOP3 / TONE	GOP3	TONE
GOP4 / nRSTO	GOP4	nRSTO
GOP5 / nIOWR	GOP5	nIOWR
GOP6	GOP6	-
GOP7 / nIORD	GOP7	nIORD
GOP8	GOP8	-
GOP9 / CLKOUT	GOP9	CLKOUT
GOP10 / FIREPULSE	GOP10	FIREPULSE
GOP11 / DRV_CSB	GOP11	DRV_CSB
GOP12 / nEDACK1	GOP12	nEDACK1
GOP13 / ECS3	GOP13	ECS3
GIOP0	Output	Input
GIOP1	Output	Input
GIOP2	Output	Input
GIOP3	Output	Input
GIOP4	Output	Input
GIOP5	Output	Input
GIOP6	Output	Input
GIOP7	Output	Input

NOTE: GIOP[4:0] is used to JTAG port in test mode.

I/O PORT MODE REGISTER

The I/O port mode register, IOPMOD, is used to configure the GIP (general input port), the GOP (general output port), and the GIOP(general in/out port).

Register	Offset Address	R/W	Description	Reset Value
IOPMOD	0x4808	R/W	I/O port mode register	0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	GIP									GOP																	GIOP				

[7:0] General In/Out mode
0 = Input mode
1 = Output mode

[21:8] General Output mode
0 = Control signal
1 = Output mode

[30:24] General Input mode
0 = Control signal
1 = Input mode

[31] USB_DMA_EN
0 = P1284 signal
1 = USB DMA signal
nINIT <==> nEXT_CDREQ
nFAULT <==> nEXT_CDACK

I/O PORT DATA REGISTER

he I/O port data register, IOPDATA, contains one-bit value for I/O ports that are configured to input mode and one-bit write value for ports that are in output mode. You can read/write the ports through the I/O port register, IOPDATA.

Register	Offset Address	R/W	Description	Reset Value
IOPDATA	0x4804	R/W	I/O port data register	Undef.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	GIP									GOP																	GIOP				

[7:0] General In/Out port

[21:8] General Output port

[30:24] General Input port - Read only

TEST CONTROL REGISTER

Test control register (TSTCON) contains 6-bits to test some functions of KS32C6400. These bits are used only for test at fabrication and are not specified in this manual. You can use the other bits as follows:

- CKOUT mode
  The CKOUT mode bit determines whether CKOUT output is divided by 2 or not.
  0 = MCLK / 2
  1 = MCLK
- Prescaler value:
  Timer 0, Timer 1, Timer 2, Watch-dog timer, and Tone generator, Line Feed Motor Timer use this prescaler value to divide MCLK.
- Bi-directional control pin

Register	Offset Address	R/W	Description	Reset Value
TSTCON	0x4800	R/W	Test control register	0x00000600

31 ... 16	15	14 13 12 11 10 9 8 7	6	5	4	3	2	1	0
	x	PRESCALER VALUE	0	x	0	0	0	0	x

[0] Chip test mode — This bit is used to control for interrupt test register(INTTEST)
0 = Normal
1 = Chip test mode

[4:1] 0 for normal operation

[5] CKOUT mode — 0 = MCLK / 2
1 = MCLK

[6] 0 for normal operation

[14:7] Prescaler value(>=1)

[15] Output value for 245CLK pin

EERAM CONTROL REGISTER

EERAMCON controls EEDATA and EECLK pins to interface a serial EEPROM. EERAMCON[0] bit responds to the EEDATA pin and EERAMCON[1] bit to the EECLK pin.

Register	Offset Address	R/W	Description	Reset Value
EERAMCON	0x5000	R/W	EERAM control register	0x00000001

31 ... 3	2	1	0
	x	x	x

[0] EERAM data

[1] EERAM clock

[2] Data direction — 0 = Input mode
1 = Output mode

JTAG TEST-LOGIC UNIT

The KS32C6400 has three modes, core test mode, the logic test mode and the MDS mode. These modes are determined by TEST1 and TEST2 pins. MDS mode supports the JTAG test logic unit. When MCU is in the MDS mode, the GIOP pins is used as a TAP (test access port). The MDS mode will be used by the ICE (Incircuit Emulator) supporting the JTAG test logic unit. The core test mode and logic test mode are used only at fabrication.

Table 14-2. MCU Operating Mode Setting

Test2	Test1	MCU State
0	0	Normal operating mode
0	1	Core test mode (only fabrication)
1	0	Logic test mode (only fabrication)
1	1	MDS mode

Table 14-3. Test Access Port Pins (MDS Mode)

TAP Pin	Share Pin	Description
TCK	GIOP[0]	Test clock input
TMS	GIOP[1]	Test mode select input
TDI	GIOP[2]	Test data input
nTRST	GIOP[3]	Test reset input
TDO	GIOP[4]	Test data output

EXTRA-OUTPUT PORT

The KS32C6400 has 13 output ports and ten additional output pins for the function blocks, which have special functions only for Ink-Jet printers. These usages are somewhat different from GOP because the pins are not designed as a dedicated output port.

To use EOPA[5:0] as an output port:

- Write the data of EOPA[5:0] pins onto the EOPA register.
- Write 0x1800 onto the EOPL (Extra-Output Port Latch Register). EOPA[5:0] pins have had the valid data.
- Do not clear EOPL register after writing 0x1800.

To use EOPB[3:0] as output port:

- Write (EOPB[3:0] << 8) | 0x8000 onto the EOPB register. The bit 14 must be “0.”

EXTRA-OUTPUT PORT LATCH REGISTER

The extra-output port latch register, EOPL, is written by “0x1800.”

Register	Offset Address	R/W	Description	Reset Value
EOPL	0x8000	R/W	Extra-output port latch register	0x00000800

EXTRA-OUTPUT PORT A REGISTER

The extra-output port A register, EOPA, contains one-bit write value to configure port to output mode.

Register	Offset Address	R/W	Description	Reset Value
EOPA	0x8004	R/W	Extra-output port A register	0x000003C0

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						RESERVED						x	x	x	x	x	x

[5:0] Extra output port

[11:6] Read only bits for chip test

15 INTERRUPT CONTROLLER

INTRODUCTION

The KS32C6400 interrupt structure has a total of 22 interrupt sources. Interrupt requests can be generated by internal function blocks and at external pins. The ARM7TDMI core recognizes two kinds of interrupts, the interrupt request (IRQ) and the fast interrupt request (FIQ). Therefore, all KS32C6400 interrupts can be categorized as either IRQ or FIQ. The KS32C6400 interrupt controller extends the number of multiple interrupt sources that can be serviced by using three special registers, INTMOD, INTPND, and INTMSK:

- INTERRUPT MODE REGISTER
  Defines the interrupt mode, IRQ or FIQ, for each interrupt source.
- INTERRUPT PENDING REGISTER
  Indicates that an interrupt requests is pending (that is, when the I-flag or F-flag is set in the program status register, PSR). This status prevents any additional interrupts from being acknowledged. When a pending bit is set, the interrupt service routine starts whenever the I-flag or F-flag is cleared to ‘0’. The service routine must clear the pending condition by writing a '1' to the appropriate pending bit.
- INTERRUPT MASK REGISTER
  Indicates that the current interrupt has been disabled if the corresponding mask bit is '0'. If an interrupt mask bit is '1', the interrupt will be serviced normally, and if the global mask bit (bit 22) is cleared, all interrupts are not serviced. However, the source's pending bit is set when the interrupt is generated even if the corresponding mask bit is '0'. After the global mask bit is set, the interrupt will be serviced.

INTERRUPT SOURCES

The 22 interrupt sources in the KS32C6400 are described as follows:

- Head DMA interrupt
- Carrier step interrupt
- Carier Position interrupt
- Line Feed Step interrupt
- General DMA 0 interrupt
- General DMA 1 interrupt
- External 0 interrupt
- Codec DMA 1 interrupt
- Host_interface interrupt
- Timer 2 interrupt
- Timer 1 interrupt
- Timer 0 interrupt
- UART channel 0 receive interrupt
- UART channel 1 receive interrupt
- UART channel 0 transmit interrupt
- UART channel 1 transmit interrupt
- UART channel 0 error interrupt
- UART channel 1 error interrupt
- External interrupt 1
- External interrupt 2
- Watch-dog timer interrupt
- Print Interrupt

INTERRUPT MODE REGISTER

Bits in the interrupt mode register, INTMOD, specify if an interrupt is to be serviced as a fast or normal interrupt.

Register	Offset Address	R/W	Description	Reset Value
INTMOD	0x4000	R/W	Interrupt mode register	0x003FFFFF

31	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

[21:0] Interrupt mode bit

Each of the 22 bits in the interrupt mode enable register, INTMOD, corresponds to an interrupt source. When the source's interrupt mode bit is set to '1' , the interrupt is processed by the ARM7TDMI core in FIQ(fast interrupt) mode. Otherwise, it is processed in IRQ mode(normal interrupt). The 22 interrupt sources are summarized as follows:

[21] HEAD DMA_TC interrupt
[20] CR motor_step/encoder interrupt
[19] CR position interrupt
[18] LF step interrupt
[17] XDMA0_TC interrupt
[16] XDMA1_TC_color chip interrupt
[15] External 0 interrupt
[14] Codec DMA 1 interrupt
[13] Host_interface terrupt
[12] Timer 0 interrupt
[11] Timer 1 interrupt
[10] Timer 2 interrupt
[9] UART channel 0 receive interrupt
[8] UART channel 1 receive interrupt
[7] UART channel 0 transmit interrupt
[6] UART channel 1 transmit interrupt
[5] UART channel 0 error interrupt
[4] UART channel 1 error interrupt
[3] External 1 interrupt
[2] External 2 interrupt
[1] Watch dog timer interrupt
[0] Print interrupt

INTERRUPT PENDING REGISTER

The interrupt pending register, INTPND, contains interrupt pending bits for each interrupt source. The INTPND has nothing to do with INTMSK. Although INTMSK forbids an interrupt request to be generated, the INTPND operates properly regardless of the INTMSK.

nEINT1,2 pins have to assert for 3 clock cycles to set the corresponding pending bit.

Register	Offset Address	R/W	Description	Reset Value
INTPND	0x4004	R/W	Interrupt pending register	0x00000000

31	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

[21:0] Interrupt pending bit

Each of the 22 bits in the interrupt pending register, INTPND, corresponds to an interrupt source. When an interrupt request is generated, it will be set by '1'.The interrupt service routine must then clear the pending condition by writing '1' to the appropriate pending bit. Only the bit written with '1' toggles from '1' to '0'. The 22 interrupt sources are summarized as follows:

[21] HEAD DMA_TC interrupt
[20] CR motor_step/encoder interrupt
[19] CR position interrupt
[18] LF step interrupt
[17] XDMA0_TC interrupt
[16] XDMA1_TC_color chip interrupt
[15] External 0 interrupt
[14] Codec DMA 1 interrupt
[13] Host_interface terrupt
[12] Timer 0 interrupt
[11] Timer 1 interrupt
[10] Timer 2 interrupt
[9] UART channel 0 receive interrupt
[8] UART channel 1 receive interrupt
[7] UART channel 0 transmit interrupt
[6] UART channel 1 transmit interrupt
[5] UART channel 0 error interrupt
[4] UART channel 1 error interrupt
[3] External 1 interrupt
[2] External 2 interrupt
[1] Watch dog timer interrupt
[0] Print interrupt

INTERRUPT MASK REGISTER

The interrupt mask register, INTMSK, contains interrupt mask bits for each interrupt source

Register	Offset Address	R/W	Description	Reset Value
INTMSK	0x4008	R/W	Interrupt mask register	0x00000000

31	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

[21:0] Interrupt mask bit

Each of the 22 bits in the interrupt pending register, INTPND, corresponds to an interrupt source. When an interrupt request is generated, it will be set by '1'. The interrupt service routine must then clear the pending condition by writing '1' to the appropriate pending bit. Only the bit written with '1' toggles from '1' to '0'. The 22 interrupt sources are summarized as follows:

[22] GLOBAL mask
[21] HEAD DMA_TC interrupt
[20] CR motor_step/encoder interrupt
[19] CR position interrupt
[18] LF step interrupt
[17] XDMA0_TC interrupt
[16] XDMA1_TC_color chip interrupt
[15] External 0 interrupt
[14] Codec DMA 1 interrupt
[13] Host_interface terrupt
[12] Timer 0 interrupt
[11] Timer 1 interrupt
[10] Timer 2 interrupt
[9] UART channel 0 receive interrupt
[8] UART channel 1 receive interrupt
[7] UART channel 0 transmit interrupt
[6] UART channel 1 transmit interrupt
[5] UART channel 0 error interrupt
[4] UART channel 1 error interrupt
[3] External 1 interrupt
[2] External 2 interrupt
[1] Watch dog timer interrupt
[0] Print interrupt

RECOGNITIONS OF THE INTERRUPT REQUESTS

The external interrupt pins (nEINT1, nEINT2) recognize interrupt requests by edge trigger. The KS32C6400 detects the down edge of nEINT1, nEINT2 and checks that the interrupt request pins are low levels during two MCLK period to confirm the interrupt request valid. The low level interval of nEINT1,2 must be longer than 3 MCLK period to confirm the validity of the interrupt request.

FIQ PRIORITY AND JUMP ADDRESS

Register	Offset Address	R/W	Description	Reset Value
FIQPRI 0	0x400C	R/W	FIQ Interrupt priority & jump address	0x00000000
FIQPRI 1	0x4010	R/W	FIQ Interrupt priority & jump address	0x00000000
FIQPRI 2	0x4014	R/W	FIQ Interrupt priority & jump address	0x00000000

31	11	10	9	8	7	6	5	4	3	2	1	0
		x	x	x	x	x	x	x	x	x	x	x

[7:0] Jump address
[10:8] FIQ priority value (0~7)

FIQ/IRQ PRIORITY AND JUMP ADDRESS

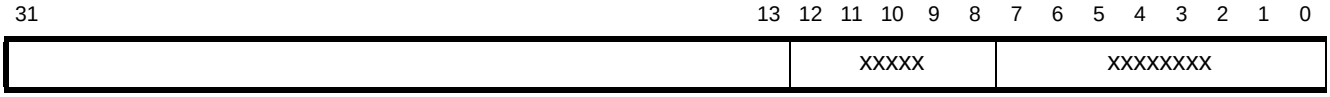
Register	Offset Address	R/W	Description	Reset Value
FIPRI 0	0x4018	R/W	FIQ/IRQ Interrupt priority & jump address	0x00000000
FIPRI 1	0x401C	R/W	FIQ/IRQ Interrupt priority & jump address	0x00000000
FIPRI 2	0x4020	R/W	FIQ/IRQ Interrupt priority & jump address	0x00000000
FIPRI 3	0x4024	R/W	FIQ/IRQ Interrupt priority & jump address	0x00000000
FIPRI 4	0x4028	R/W	FIQ/IRQ Interrupt priority & jump address	0x00000000

31	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		x	x	x	x	x	x	x	x	x	x	x	x	x

[7:0] Jump address
[12:8] FIQ/IRQ priority value (Do not setting by zero)

IRQ PRIORITY AND JUMP ADDRESS

Register	Offset Address	R/W	Description	Reset Value
IRQPRI 0	0x402C	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 1	0x4030	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 2	0x4034	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 3	0x4038	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 4	0x403C	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 5	0x4040	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 6	0x4044	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 7	0x4048	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 8	0x404C	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 9	0x4050	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 10	0x4054	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 11	0x4058	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 12	0x405C	R/W	IRQ Interrupt priority & jump address	0x00000000
IRQPRI 13	0x4060	R/W	IRQ Interrupt priority & jump address	0x00000000

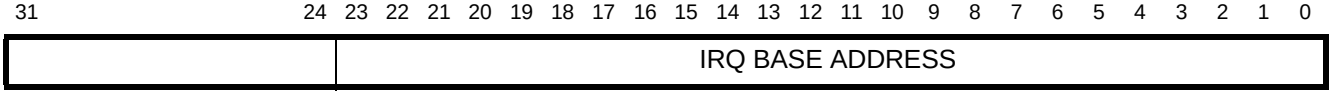


[7:0] Jump address

[12:8] IRQ priority value (Do not setting by zero)

IRQ BASE ADDRESS

Register	Offset Address	R/W	Description	Reset Value
IRQBASE	0x4064	R/W	IRQ base address	0x00000000



[23:0] IRQ base address

FIQ BASE ADDRESS

Register	Offset Address	R/W	Description	Reset Value
FIQBASE	0x4068	R/W	FIQ base address	0x00000000

31	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		FIQ BASE ADDRESS																							

[23:0] FIQ base address

INTERRUPT TEST REGISTER

Register	Offset Address	R/W	Description	Reset Value
INTTEST	0x406C	W	Interrupt generate register for test	0xFFFFFFFF

31	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

[21:0] Interrupt generate

If chip test mode bit(TSTCON) is “1”, this register is valid.
1 = Normal
0 = Interrupt generate

[21] HEAD DMA_TC interrupt
[20] CR motor_step/encoder interrupt
[19] CR position interrupt
[18] LF step interrupt
[17] XDMA0_TC interrupt
[16] XDMA1_TC_color chip interrupt
[15] External 0 interrupt
[14] Codec DMA 1 interrupt
[13] Host_interface terrupt
[12] Timer 0 interrupt
[11] Timer 1 interrupt
[10] Timer 2 interrupt
[9] UART channel 0 receive interrupt
[8] UART channel 1 receive interrupt
[7] UART channel 0 transmit interrupt
[6] UART channel 1 transmit interrupt
[5] UART channel 0 error interrupt
[4] UART channel 1 error interrupt
[3] External 1 interrupt
[2] External 2 interrupt
[1] Watch dog timer interrupt
[0] Print interrupt

IRQ INTERRUPT JUMP ADDRESS

Register	Offset Address	R/W	Description	Reset Value
IRQJPADDR	0x4070	R	IRQ Interrupt jump address	0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

FIQ INTERRUPT JUMP ADDRESS

Register	Offset Address	R/W	Description	Reset Value
FIQJPADDR	0x4074	R	FIQ Interrupt jump address	0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

16 LF MOTOR

INTRODUCTION

This module performs the following functions:

- Step interrupt generation for driving line feed motor.
- 14bit timer for step interrupt which controls the change of drive signal for line feed motor using selectable clock.
- Phase is written by software or hardware.

NOTE

1. When the timer is enabled, it begins to decrease from the base value.
2. When the timer expires, the associated interrupt is generated and the base value is reload and the timer continues to decrease.
3. If new value is loaded in this register before the timer is expired, the timer will keeping counting with previous value untill the timer expired, and load the new base value, start counting again.

LINE FEED MOTOR CONTROL REGISTER

This register selects the phase is written by software or hardware, direction, interrupt request generation, and clock select.

Registers	Offset Address	R/W	Description	Reset Value
LFCR	0x8000	R/W	Line Feed motor control register	0x00000800

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		xx		x	x	x	x	x	x	x	x	x	x	x	CLK		x

[0] Timer start
0 = stop timer
1 = start timer

[2:1] Clock select
00 = MCLK / (prescaler + 1) / 8
01 = MCLK / (prescaler + 1) / 16
10 = MCLK / (prescaler + 1) / 32
11 = MCLK / (prescaler + 1) / 64

[3] Interrupt request at each step
0 = disable
1 = enable

[4] Write strobe(write only)
0 = No operation
1 = Write strobe

[5] Step write enable
0 = disable
1 = enable

[6] Step
0 = half step
1 = full step

[7] Direction
0 = up
1 = down

[8] Chop enable
0 = disable
1 = enable

[9] Auto mode enable
0 = Phases have to be written by software
1 = Phases are generated automaticaly

[10] Motor selection
0 = Bi-polar
1 = Uni-polar

[11] Initialize drive state(write only)
0 = initialize
1 = normal

[12] Software latch strobe (write only)
0 = No operation
1 = Latch strobe

[13] Interrupt latch enable
0 = disable
1 = enable

[15:14] Phase
00 = Full step
01 = Half step
1x = Quarter step

LINE FEED MOTOR PHASE CONTROL REGISTER

Registers	Offset Address	R/W	Description	Reset Value
LFPCR	0x8004	R/W	Line Feed motor phase control register	0x000003C0

31	11	10	9	8	7 6	5	4	3 2 1 0
	x	x	x	x	xx	x	x	PHASE

[3:0] PHASE
Uni-polar Mode : Phase Az, A, Bz, B
Bi-polar Mode : Current IA1, IA0, IB1, IB0
These signals are used in S/W mode.

[4] LF Motor control 1 / PHA
0 = Low
1 = High

[5] LF Motor control 2 / PHB
0 = Low
1 = High

[7:6] PHB_IB0, PHBZ_IB1
- Default Phase[1:0] (when write in Bi-polar mode)
  00 = AB 01 = AzB
  10 = AzBz 11 = ABz
- PHB, PHBZ (when read in Uni-polar mode)
- IB1, IB0 (when read in Bi-polar mode)

[8] PHAZ_IA1 (read only)
0 = Low
1 = High

[9] PHA_IA0 (read only)
0 = Low
1 = High

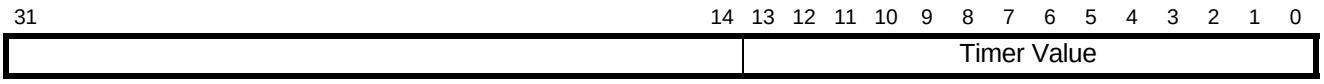
[10] LFPHASEB (read only)
0 = Low
1 = High

[11] LFCON_PHA (read only)
0 = Low
1 = High

LINE FEED TIMER REGISTER

These 14-bit timer is used to generate the Line Feed motor's phase which is driven by software or hardware according to line feed motor control register.

Registers	Offset Address	R/W	Description	Reset Value
LFTBR	0x8008	R/W	LF motor phase control register	Undef.
LFTOR	0x800C	R	LF motor timer observation register	Undef.
LFTCBR	0x8010	R/W	LF motor timer compare base register	0x00000000
LFTCOR	0x8014	R	LF motor timer compare obseration register	0x00000000



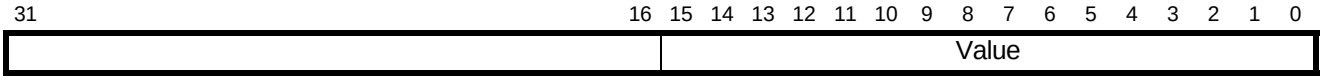
[13:0] Timer Value This time is used to generate line feed motor’s phase to phase interval.

You have to set by zero for lfcr[12] and lfpcr[5:0] before LF Motor Bi-polar operate.

LFCON EACH CONTROL REGISTER

This register is used to set active current level of each step in Bi-polar and mode holding value.

Registers	Offset Address	R/W	Description	Reset Value
LFCON	0x8018	R/W	LF step each control register	0x00000000



[15:0] Value LFCON each control register

17 CR CONTROL

INTRODUCTION

This module is configured as following block:

- Basic Timer using MCLK clock is 16 bit down counter.
- Prestep timer using 19200/9600 ppi clock is 10bit down-counter.
- Phase and current control signals generation for step motor of bi-polar type.
- Filter for photo sensor input, position counting strobe and direction generation for DC motor.
- Encoder cycle counter of 20bit up-counter and latch with MCLK clock to calculate the cycle of photo sensor input in DC motor mode.
- Interrupt interval counter of 16 bit up-counter and latch with MCLK/32 clock to calculate the interval of Carrier Interrupt in DC motor mode.

This module performs the following functions in Step Motor Mode:

- Basic timer generates the basic pulse of 19200/9600 pulse per inch to control the state and position of carrier step motor and to generate fire strobe to control the printhead.
- Prestep timer is used to generate carrier step pulse to control the change of output state signals which are phase and current control signals for carrier motor driver and carrier step interrupt according to carrier motor step rate.
- State control block is used to generate two phase and four current control signals for every step interrupt according to setting of motor direction and state mode.

This module performs the following functions in DC Motor Mode:

- Filter block is used to protect false information by the noise onto input signals from photo sensor.
- Encoder counter is used to calculate and store the cycle time of preceding input from photo sensor. this cycle time is used for base value of basic timer which generate the basic pulse of 2400 PPI for fire strobe control in DC motor mode, and this time value can be read by CPU.
- Interrupt counter is used to calculate and store the interval time of DC motor interrupt. If this counter overflow before the next interrupt have been observed, an interrupt will be issued, counter will be zeroed.
- In DC motor mode, Prestep timer is used to set the number of rising edge on preceding input from photo sensor to issue an interrupt for DC motor position control.

NOTE

1. Writing the value of basic timer base register 1 must be preceded that of basic timer base register 2 after the reset is done.
2. When the basic timer based on one of base registers is running, if the next base value is written to the other base register, counter will keep counting with next value. If the next base value is not written to the other register any more, counter will repeat counting with current base value until the timer is disabled.
3. In order to set DC motor mode, bit 9 of CMCR have to be set "1".
4. For the step motor mode, CMCR[11:4] have to set zero.

CARRIER MOTOR CONTROL REGISTER

This register determines interrupt request is generated or not, and enable or disable prestep timer and basic timer.

Registers	Offset Address	R/W	Description	Reset Value
CMCR	0x9000	R/W	Carrier motor control register	0x00000004

31	12	11	10	9	8	7	6	5	4	3	2	1	0
		X	X	X	X	X	X	X	X	X	X	X	X

[0] Basic timer enable
0 = disable
1 = enable

[1] Preset timer enable
0 = disable
1 = enable

[2] Carrier step interrupt enable
0 = disable
1 = enable

[3] Not used

[4] DC Motor Output enable
0 = disable
1 = enable

[5] DC Motor Output Selection/ Direction
0 = DCOUT1 is enabled / DIR=0
1 = DCOUT2 is enabled / DIR=1

[6] PWM Timer RUN
0 = Stop
1 = RUN

[7] PWM Mode Selection
0 = 2 PWM Signal mode
1 = 1 PWM and Direction Signal mode

[8] Preceding Encoder Input Selection
0 = CHX
1 = CHY

[9] Motor Type Selection
0 = Step Motor
1 = DC Motor

[10] Head Type Selection in DC Mode
0 = 208 Nozzle Head
1 = 56 Nozzle Head

[11] Overflow Interrupt Enable
0 = disable
1 = enable DC_Interrupt causing overflow of Interrup Interval counter

BASIC TIMER BASE REGISTER

There are two basic timers, BTB1R, BTB2R, at first the timer starts with the value of BTB1R, after down-count stops the timer starts with the value of BTB2R at only BTB2R is written to a new value, otherwise, the timer starts with current base timer value(repeat mode).

Registers	Offset Address	R/W	Description	Reset Value
BTB1R	0x9004	R/W	Basic timer base register 1	Undef.
BTB2R	0x9008	R/W	Basic timer base register 2	Undef.

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		COUNT VALUE															

[15:0] COUNT VALUE Base timer 1/2 Count value

PRESTEP TIMER BASE REGISTER

Registers	Offset Address	R/W	Description	Reset Value
PSTBR	0x900C	R/W	Prestep timer base register	Undef.

31	10	9	8	7	6	5	4	3	2	1	0
		COUNT VALUE									

[9:0] COUNT VALUE Prestep Count value

CR STATE CONTROL REGISTER

This register can control to generate two phase lines an four current control lines to drive a bipolar stepping motor. Eight output combinations are sequentially presented on these six lines.

Registers	Offset Address	R/W	Description	Reset Value
CRSCR	0x9010	R/W	CR state control register	0x0000E03A

31 ... 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	x	x	x	x	IA		IB		x	x	PHASE STATUS					

[5:0] Phase status
- [0] IB0
- [1] IB1
- [2] IA0
- [3] IA1
- [4] PHASE B
- [5] PHASE A

[6] Write strobe(write only)
- 0 = No operation
- 1 = Write strobe

[7] Write enable
- 0 = disable
- 1 = enable

[9:8] IB — Phase B current control(IB0,IB1)

[11:10] IA — Phase A current control(IA0,IA1)

[12] Direction
- 0 = up
- 1 = down

[13] Step
- 0 = Half step
- 1 = Full step

[14] Phase change enable
- 0 = disable
- 1 = enable

[15] Initial drive state control(write only)
- 0 = Initialize
- 1 = Normal

CRSREG EACH CONTROL REGISTER

This register is used to set active current level for each step in Bi-polar mode. You can refer phase state and current table in next page for detail current level of each step.

Registers	Offset Address	R/W	Description	Reset Value
CRSREG	0x9030	R/W	CR step each control register	0x00000000

31 ... 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	VALUE															

[15:0] VALUE — CRSREG each control register

Phase State and Current Table for Full/Half/Quarter Step Mode

Direction	Phase State	Current State & Level					Each Control Register	Holding Register = 1	Remark
		IA1	IA0	IB1	IB0	IA/IB %			
	AB	0 0	0 1	0 0	0 1	100/100% 66/66%	0 1	33/33%	
	AB	1	0	0 0	0 1	33/100% 33/66%	0 1	33/33%	
CW CCW	(A)B (A_Z)B	1	1	0 0	0 1	0/100% 0/66%	0 1	0/33%	
	A_ZB	1	0	0 0	0 1	33/100% 33/66%	0 1	33/33%	
	A_ZB	0 0	0 1	0 0	0 1	100/100% 66/66%	0 1	33/33%	
	A_ZB	0 0	0 1	1	0	100/33% 66/33%	0 1	33/33%	
CW CCW	A_Z(B) A_Z(B_Z)	0 0	0 1	1	1	100/0% 66/0%	0 1	33/0%	
	A_ZB_Z	0 0	0 1	1	0	100/33% 66/33%	0 1	33/33%	
	A_ZB_Z	0 0	0 1	0 0	0 1	100/100% 66/66%	0 1	33/33%	
	A_ZB_Z	1	0	0 0	0 1	33/100% 33/66%	0 1	33/33%	
CW CCW	(A_Z)B_Z (A)B_Z	1	1	0 0	0 1	0/100% 0/66%	0 1	0/33%	
	AB_Z	1	0	0 0	0 1	33/100% 33/66%	0 1	33/33%	
	AB_Z	0 0	0 1	0 0	0 1	100/100% 66/66%	0 1	33/33%	
	AB_Z	0 0	0 1	1	0	100/33% 66/33%	0 1	33/33%	
CW CCW	A(B_Z) A(B)	0 0	0 1	1	1	100/0% 66/0%	0 1	33/0%	
	AB	0 0	0 1	1	0	100/33% 66/33%	0 1	33/33%	

PWM TIMER

LOGIC CONFIGURATION

PWM block consists of Cycle_Time base register, On_Time base register, Counter observation register and 16bit down-counter.

FUNCTION

- Duration of PWM output signal and On/Off time in the this duration is decided by Cycle_Time base value and On_Time base value. If On Time base value is greater or equal than Cycle Time base value, PWM output signal maintain On status consequently.
- 16-bit down counter is determined by RUN or STOP of CMCR[6].
- Oeration and Output Status of PWM Block reguarding to bit 4, 5, 7 of CMCR as following table:

Table 1-1. Operation and Output Status of PWM Block

CMCR	Description	DC motor Control Signal Output Status			
CMCR[4]	DC Motor Output Enable	Enables the PWM outputs			
CMCR[7]	PWM mode selection	0		1	
CMCR[5]	DC Motor Direction / Output Selection	0	1	x	x
DC_CRIA0 Pin		PWM Signal	0	PWM Signal	
DC_CRIA1 Pin		0	PWM Signal	Direction	

PWM COUNTER BASE REGISTER AND OBSERVATION REGISTER

Registers	Offset Address	R/W	Description	Reset Value
PWMOBS	0x9014	R	PWM counter observation register	0x00000000
PWMCYL	0x9018	R/W	PWM cycle time base register	0x00000000
PWMONT	0x901C	R/W	PWM on time base register	0x00000000

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		COUNT VALUE															

[15:0] COUNT VALUE PWM Cycle, On time base value

NOTE

Real Cycle Time = (base value +1) x 1/MCLK
Real On Time = (base value + 1) x 1/MCLK

This block only operates on DC mode.

ENCODER COUNTER

LOGIC CONFIGURATION

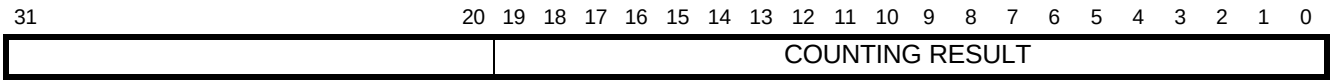
This block consists of 20-bit register which stores 20-bit up counter and counting results value.

FUNCTION

- Counting input duration of Photo sensor(Encoder Sensor) and store duration value in register.
- This duration value is used to base value of calculation of Fire Window Time and Fire Strobe according to setting of Fire DPI.

COUNTERING RESULT REGISTER AND OBSERVATION REGISTER

Registers	Offset Address	R/W	Description	Reset Value
ECDTIM	0x9020	R	Encoder counter observation register	0x00000000
ECDVAL	0x9024	R	Encoder cycle value register	0x00000000



[19:0] COUNTING RESULT Encoder Cycle Value

NOTE: This block only operates on DC mode.

INTERRUPT INTERVAL COUNTER

LOGIC CONFIGURATION

This block consists of 16-bit register which stores 16-bit up conter and counting results.

FUNCTION

- This logic operates since first DC motor interrupt is generates.
- It stores counting interval value between DC motor interrupt and next interrupt in register.
- If counter overflow generate before next interrupt generates when CMCR[11] is '1', DC motor interrupt is generated and restart after clear this counter and pre step timer
- If rising edge of Photo(Encoder) sensor input as Pre step timer count value is generated in DC mode, DC motor interrupt is generated.

COUNTERING RESULT REGISTER AND OBSERVATION REGISTER

Registers	Offset Address	R/W	Description	Reset Value
INTTIM	0x9028	R	Interval counter observation register	0x00000000
INTVAL	0x902C	R	Interrupt interval value register	0x00000000

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		COUNT VALUE															

[15:0] COUNTING RESULT Interrupt interval value

18 CR FIRE

INTRODUCTION

This module performs the following funtions:

- Count and control the position of carrier motor.
- Fire strobe and start signal generation.
- 16bit counter for the position of carrier motor.
- 16bit print slice counter for counting fire strobe.
- 6bit prescaler for the clock of carrier position.
- 8bit prescaler for the clock of fire strobe.

NOTE

1. This block is responsible for positioning the printhead and regulating the printhead fire strobe timing.
2. Two conditions must be met before the fire strobe logic will be activated.
   First, the print slice count must be greater than zero.
   Second, the position counter must be equal to the print start position.
3. When the start position is reached, the fire logic is enabled, and the first fire strobe is generated. Each fire strobe decrease the slice count by one. When the slice count reaches to zero, the fire logic is disabled.
4. For step motor mode, the cycle of fire strobe is decided by setting only base value of fire prescaler. (Fire DPI = 19200 / Fire prescaler) For DC motor mode, the cycle of fire strobe is decided by setting base value of fire prescaler and DPI mode setting bit of PFCR.

POSITION AND FIRE CONTROL REGISTER

Registers	Offset Address	R/W	Description	Reset Value
PFCR	0x9820	R/W	Position & Fire control register	0x00008090

31	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PRESCALER							PRESCALER									x	x	MODE			x

[0] Position prescaler enable
0 = disable
1 = enable

[3:1] DPI mode(Only for DC_Mode)
000 = 150dpi
001 = 300dpi
010 = 600dpi
011 = 1200dpi
100 = 2400dpi
Others = 150dpi

[4] Position count reset(write only)
0 = Reset
1 = Normal operation

[5] Window time base select
0 = Time base is written by software
1 = Time base is written by hardware

[6] Not used

[14:7] Prescaler
This 8bit prescaler is for the fire strobe.

[20:15] Prescaler
This 6bit prescaler is for carrier position.

CR POSITION AND FIRE CONTROL REGISTER

There are four registers in this block, Carrier Position Count Register(CPCR), Print Start Position Register (PSPR), Print Slice Counter Register(PSCR), and Position Interrupt Register(PIR).

CPCR : The carrier position is updated based on the carrier movement of 1/600 inch.
PSPR : The fire strobe control logic requires two conditions to be met before it will generate fire strobe to print logic.
1) A non-zero value must be loaded into the print slice counter register.
2) The carrier position must match the value in the print start position register.
3) Once the two requirements have been met, the logic will begin producing fire strobes after 1/75 inch.

PSCR : This value is decreased once for each fire strobe that is generated.
PIR : When outputs of carrier position counter become same as the value of this register, position interrupt request is occured.

Registers	Offset Address	R/W	Description	Reset Value
CPCR	0x9824	R/W	Carrier position counter register	0x00000000
PSPR	0x9828	R/W	Print start position register	0x0000FFFF
PSCR	0x982C	R/W	Print slice counter register	0x00000000
PIR	0x9830	R/W	Position interrupt register	0x0000FFFF

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		COUNT VALUE															

[15:0] Count value of each register

EXAMPLE OF POSITION DPI SETTING FOR STEP MOTOR MODE

Position DPI	Prescaler Value (PFCR[20:15])
300 DPI	64 (19200/300)
600 DPI	32 (19200/600)
1200 DPI	16 (19200/1200)

EXAMPLE OF FIRE DPI SETTING

FIRE DPI	DC Motor Mode		Step Motor Mode
	Prescaler Value (PFCR[14:7])	DPI Mode Setting Value	Prescaler Value (PFCR[14:7])
150 DPI	16 (2400/150)	000, Other Case	128 (19200/150)
300 DPI	8 (2400/300)	001	64 (19200/300)
600 DPI	4 (2400/600)	010	32 (19200/600)
1200 DPI	2 (2400/1200)	011	16 (19200/1200)
2400 DPI	1 (2400/2400)	011	8 (19200/2400)

19 PRINT HEAD

INTRODUCTION

This module performs the following funtions:

- Fire pulse generation.
- DMA request for reading data.
- Three 32-bit dot counters for color.
- One 32-bit dot counter for mono.
- Fire strobe delay for horizontal alignment of dot.
- 8-bit decrement timer for the width of the fire enable pulse of printhead logic using MCLK.
- 10-bit decrement timer for the width of the fire group window of printhead logic using MCLK.
- Four 12-bit timers for the fire strobe delay using selectable clock.
  (clock = main clock / 1, 2, 4, or 8)
- 4-bit decrement counter for Td delay.
- 6-bit pre-heat pulse timer.
- 6-bit pre-heat delay timer.

NOTE

1. The printhead logic is responsible for receiving image data, conditioning the data for print and routing the data to the printhead in the proper sequence. The nature of the printhead design is the primary motivation for the structure of the printhead firing logic.
2. The printhead contains 208 nozzles arranged in two columns that are divided into sixteen groups containging 13 nozzles each. The printhead is configured such that only one nozzle from each group may be fired at any one time. This necessitates a sequential firing scheme passing through each of the 13-nozzle group one at a time firing a maximum of sixteen nozzles.
3. Many of the critical timing relationships for the printhead firing are controlled by Carrier motor logic. It is imperative that printhead motion and nozzle firing be directly linked to produce the desired print output characteristics. Additional software control is provided to aid in print alignment.
4. Data for the printhead logic is received from memory via the HDMA pair or directly from the JupiterII. During a print fire cycle, the logic will issue a data request for 3, 4, 12, or 13 halfwords of data transferred per print slice.

Head Control Register			Head Type	Number of Data
PHCR[8]	PHCR[7]	PHCR[11]		
0	0	0	DH, Mono Head (208 Nozzle)	13 Halfword
0	1	0	DH, Color Head (192 Nozzle)	12 Halfword
1	0	0	SH, Mono Head (56 Nozzle)	4 Halfword
1	1	0	SH, Color Head (48 Nozzle)	3 Halfword

PRINT HEAD CONTROL REGISTER

Registers	Offset Address	R/W	Description	Reset Value
PHCR	0xA000	R/W	Print head control register	0x00000000

31 26	25	24	23 22 21 20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3 2 1 0
	x	x	ADDRESS	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	DITHER

[3:0] Dither count bit

[4] Decrement through enable group
0 = Up
1 = Down

[5] Black dot counter select
0 = Disable
1 = Enable

[6] Color dot counter select
0 = Disable
1 = Enable

[7] Color Head select
0 = Black
1 = Color

[8] Head Type select
0 = 208/192 nozzle head
1 = 56/48 nozzle head

[9] Consecutive dot eliminator
0 = Disable
1 = Enable

[10] Top nozzle group select
0 = Right
1 = Left

[11] Vertical 300dpi mode
0 = Halfword
1 = Byte

[12] Horizontal 300dpi mode
0 = Disable
1 = Enable

[13] Decompression mode
0 = Disable
1 = Enable

[14] Data select
0 = by software
1 = by hardware

[15] Simulation test control (HDMA request generation)
0 = Disable
1 = Enable

[16] Perform a fire cycle (write only)
0 = Disable
1 = Enable

[17] Perform a data cycle (write only)
0 = Disable
1 = Enable

[19:18] Clock select
00 = MCLK/1 01 = MCLK /2
10 = MCLK/4 11 = MCLK/8

[23:20] Address line (for the nozzle)

[24] Address line (by software)
0 = Disable
1 = Enable

[25] Current mode
0 = Printing
1 = Scanning

FIRE ENABLE TIMER/OBSERVATION REGISTER

This 8-bit timer is used for fire enable duration counter value. The observation register is read-only which is current value.

Registers	Offset Address	R/W	Description	Reset Value
FETR	0xA004	R/W	Fire enable timer register	0x00000000
FETOR	0xA008	R	Fire enable timer observation register	0x00000000

31	8	7	6	5	4	3	2	1	0
	COUNTER VALUE								

[7:0] COUNTER VALUE This data specifies the amount of counter

FIRE WINDOW TIMER/OBSERVATION REGISTER

This 10-bit timer is used for fire window enable duration counter value. The observation register is read-only which is current value.

Registers	Offset Address	R/W	Description	Reset Value
FWTR	0xA00C	R/W	Fire window timer register	0x00000000
FWTOR	0xA010	R	Fire window timer observation register	0x00000000

31	10	9	8	7	6	5	4	3	2	1	0
	COUNTER VALUE										

[9:0] COUNTER VALUE This data specifies the amount of counter

FIRE STROBE DELAY TIMER/OBSERVATION REGISTER

This 12-bit timer is used for fire strobe delay duration counter value. The observation register is read-only which is current value.
The printhead logic contains four 12-bit timers that are used to delay the fire strobes from the carrier motor logic before sending them to the printhead drivers. The timers alternate for each fire strobe.
While one timer is delaying a pulse, the other will be waiting for the next pulse. When the timer decrease to zero, it passes the pulse to the fire strobe state machine. If the timer is still running when four slices have passed, the next fire strobe will be generated and the timer will be forced to reload. This will present the loss of data from excessive delays.

Registers	Offset Address	R/W	Description	Reset Value
FSDTR	0xA014	R/W	Fire strobe delay timer register	0x00000000
FSDT0OR	0xA018	R	Fire delay strobe timer 0 observation register	0x00000000
FSDT1OR	0xA01C	R	Fire delay strobe timer 1 observation register	0x00000000
FSDT2OR	0xA020	R	Fire delay strobe timer 2 observation register	0x00000000
FSDT3OR	0xA024	R	Fire delay strobe timer 3 observation register	0x00000000

31	12	11	10	9	8	7	6	5	4	3	2	1	0
	COUNTER VALUE												

[11:0] COUNTER VALUE This data specifies the amount of counter

PRE-HEAT PULSE TIMER/OBSERVATION REGSITER

This 6-bit timer is used for pre-heat pulse enable duration counter value. The observation register is read-only which is current value.

Registers	Offset Address	R/W	Description	Reset Value
PHPTR	0xA028	R/W	Pre-heat pulse timer register	0x00000000
PHPTOR	0xA02C	R	Pre-heat pulse timer observation register	0x00000000

31	6	5	4	3	2	1	0
	COUNTER VALUE						

[5:0] COUNTER VALUE This data specifies the amount of counter

PRE-HEAT DELAY TIMER/OBSERVATION REGSITER

This 6-bit timer is used for pre-heat delay enable duration counter value. The observation register is read-only which is current value.

Registers	Offset Address	R/W	Description	Reset Value
PHDTR	0xA030	R/W	Pre-heat delay timer register	0x00000000
PHDTOR	0xA034	R	Pre-heat delay timer observation register	0x00000000

31	6	5	4	3	2	1	0
	COUNTER VALUE						

[5:0] COUNTER VALUE This data specifies the amount of counter

PRINT HEAD OBSERVATION REGSITER

Registers	Offset Address	R/W	Description	Reset Value
PHOR	0xA038	R	PrintHead observation register	0x00000000

31	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SUM						DELAY				NUMBER				DITHER				HDMA				FIRE		DATA				

[3:0] Data State This indicates the printhead data state machine value

[6:4] Fire State This indicates the printhead fire state machine value

[10:7] HDMA state This indicates the printhead HDMA state machine value

[14:11] Dither Count

[18:15] Number of current printhead data from HDMA

[22:19] Front/Back end delay counter value

[27:23] Sum of the number of output enable signals.

Td DELAY COUNTER REGSITER

This 4-bit timer is used for front/back end delay duration counter value.

Registers	Offset Address	R/W	Description	Reset Value
TDCR	0xA03C	R/W	Td delay counter register	0x00000000

31	8	7	6	5	4	3	2	1	0
					FRONT			BACK	

[3:0] Counter 1 Value This data specifies the amount of back-end delay counter

[7:4] Counter 2 Value This data specifies the amount of back-end delay counter

PRINT HEAD DATA WORD REGSITER

Registers	Offset Address	R/W	Description	Reset Value
PHDW0R	0xA040	R/W	PrintHead data word 0 register	0x00000000
PHDW1R	0xA044	R/W	PrintHead data word 1 register	0x00000000
PHDW2R	0xA048	R/W	PrintHead data word 2 register	0x00000000
PHDW3R	0xA04C	R/W	PrintHead data word 3 register	0x00000000
PHDW4R	0xA050	R/W	PrintHead data word 4 register	0x00000000
PHDW5R	0xA054	R/W	PrintHead data word 5 register	0x00000000
PHDW6R	0xA058	R/W	PrintHead data word 6 register	0x00000000
PHDW7R	0xA05C	R/W	PrintHead data word 7 register	0x00000000
PHDW8R	0xA060	R/W	PrintHead data word 8 register	0x00000000
PHDW9R	0xA064	R/W	PrintHead data word 9 register	0x00000000
PHDW10R	0xA068	R/W	PrintHead data word 10 register	0x00000000
PHDW11R	0xA06C	R/W	PrintHead data word 11 register	0x00000000
PHDW12R	0xA070	R/W	PrintHead data word 12 register	0x00000000

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		PRINTHEAD DATA WORD															

[15:0] Printhead data word

DOT COUNTER REGSITER

Registers	Offset Address	R/W	Description	Reset Value
DCBR	0xA074	R/W	Dot counter black register	0x00000000
DCYR	0xA078	R/W	Dot counter yellow register	0x00000000
DCCR	0xA07C	R/W	Dot counter cyan register	0x00000000
DCMR	0xA080	R/W	Dot counter magenta register	0x00000000

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DOT COUNTER VALUE																	

[31:0] Dot Counter Value — This data specifies the amount of counter of black for mono head and yellow, cyan and magenta for color head.

DOT COUNTER CONTROL OBSERVATION REGSITER

Registers	Offset Address	R/W	Description	Reset Value
DCCOR	0xA084	R/W	Dot counter control observation register	0x00000000

31	9	8	7	6	5	4	3	2	1	0
		Magenta			Cyan			Yellow		

[2:0] Yellow Dots — This data specifies the amount of yellow dots per a fire pulse

[5:3] Cyan Dots — This data specifies the amount of cyan dots per a fire pulse

[8:6] Magenta Dots — This data specifies the amount of magenta dots per a fire pulse

BLANK PAGE

20 HDMA

INTRODUCTION

This module is used to transfer head data from Memory to head data register by DMA with or without match function.

HEAD DMA CONTROL REGISTER

This DMA has a kind of DMA operation under the control of print module. HDMA reads from memory, write to head. HDMA can transfer data by bytes/half-words. The transfer size is decided by setting the head control register.

Registers	Offset Address	R/W	Description	Reset Value
HDCON	0xD000	R/W	Head DMA control register	0x00000000

31	25	24	17	16	12	11	10	9	8	7	6	5	4	3	2	1	0
		X	-	X	-	X	X	X	X	X	X		X	X	X	X	X

[0] Run eable/disable

When you set this bit to '1', HDMA operation starts. To stop HDMA,you must clear this bit to '0'. If you control this bit only, 0xc010 address will be used.(if 0xc010 address is used, other value will not be changed.)

0 = Disable HDMA operation
1 = Enable HDMA operation

[1] BUSY status

When HDMA starts, this read-only status bit is automatically set to '1'. When it is '0', HDMAis an idle status.

0 = Idle HDMA
1 = Active HDMA

[2] Source addr direction

Only one source can initiate a HDMA operation. If this bit is set, the source address will be decreased. If this bit is cleared, the source address will be increased.

0 = Increase source address
1 = Decrease source address

[3] Source addr fix

This bit determines whether the source address will be changed or not during a HDMAoperation. This feature is used when transferring data from a single source to multiple destinations.

0 = Increase/Decrease source address
1 = Do not change source address(fix)

[4] Reset

If this bit is set to '1', then HDMAcontrol register value will be initialized and then after this bit is cleared to '0', you can specify other control value

0 = Normal operation
1 = Initialize control register

[6] Match Pend Status

If the value of the source address register(HDSAR) and the value of the match address register(HDMAR) are matched, Match pend status bit is set.. If you would like to clear the status bit, write zero value.

0 = Not match
1 = Match

[7] Match Interrupt Pending Enable

This bit determines whether the interrupt by match of source/match address register occurs or not. In case of match, HDMA operates until source address will be match address.

0 = Enable match interrupt
1 = Disable match interrupt

[8] HDMA Interrupt enable	If this bit is set to '1', a interrupt is generated when HDMA operation stop, Tx counter is finished or match occur. If this bit is '0', all of these interrupt do not occur. 0 = Do not generate stop interrupt and Match interrupt 1 = Generate stop interrupt and Match interrupt when HDMA stops
[9] Auto Load	This bit should be enabled for source address register's parallel load. 0 = Do not load parallel load 1 = Load parallel load of counter register
[10] Alternate Enable	This bit determines to alternate register banks. 0 = Can not operates alternation 1 = Can operates alternation between bank 0 and bank 1
[11] Current Queuing Bank Selection	This bit indicates whether current queuing bank is 0 or 1. You can set/ clear the Queuing bit selection. 0 = Queing to bank 0 1 = Queing to bank 1
[16] Queuing 0 enable	If this bit is set and HDCON[10] is set, HDMA alternates bank 0. 0 = Disable queuing 0 1 = Enable queuing 1
[24] Queuing 1 enable	If this bit is set and HDCON[10] is set, HDMA alternates bank 1. 0 = Disable queuing 0 1 = Enable queuing 1

HDMA SOURCE ADDRESS REGISTER

These registers contain the 28-bit source/destination address for HDMAchannel. Depending on the settings you make to the HDMA control register(HDCON), theses adr will be fixed or increased or decreased.

Registers	Offset Address	R/W	Description	Reset Value
HDSAR	0xD004	R/W	HDMA source address register	Undef.

31	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		SOURCE ADDRESS																											

[27:0] Source Address

HDMA TRANSFER COUNT REGISTER

This register contains the 24-bit current count value of the number of HDMA transfers completed for HDMA. This count value is decreased by 1 while one DMA operation is completed regardless of transfer width.

Registers	Offset Address	R/W	Description	Reset Value
HDTCR	0xD00C	R/W	HDMA transfer count register	Undef.

31	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TRANSFER COUNT																								

[23:0] Transfer Count

HDMA SOURCE/MATCH ADDRESS REGISTER

These registers contain the 28-bit source/destination address for HDMA channel. Depending on the settings you make to the HDMA control register(HDCON), theses adr will be fixed or increased or decreased.

Registers	Offset Address	R/W	Description	Reset Value
HDSAR0	0xD014	R/W	HDMA source address register 0	Undef.
HDMAR0	0xD018	R/W	HDMA match address register 0	Undef.
HDSAR1	0xD01C	R/W	HDMA source address register 1	Undef.
HDMAR1	0xD020	R/W	HDMA match address register 1	Undef.

31	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		SOURCE/MATCH ADDRESS																											

[27:0] Source/Match Address

Examples of setting registers to use match and queuing funtion of HDMA

1. Case 1 : Bank0 -> Bank1 -> Bank 0 -> Bank1 -> Bank 0
   1) Set HDSAR0, HDMAR0, HDSAR1, HDMAR1, HDTCR
   2) Set HDCON with 0x1010781
      (Match Interrupt Pending Enable, HDMA Interrupt Enable, Auto Load Enable, Aternate Enable, Select bank0 for current queuing bank, Enable Queuing Enable Bit 0/1, Run HDMA)

2. Case 2 : Bank1 -> Bank 0 -> Bank1 -> Bank 0 -> Bank1
   1) Set HDSAR0, HDMAR0, HDSAR1, HDMAR1, HDTCR
   2) Set HDCON with 0x1010f81
      (Match Interrupt Pending Enable, HDMA Interrupt Enable, Auto Load Enable, Aternate Enable, Select bank1 for current queuing bank, Enable Queuing Enable Bit 0/1, Run HDMA)

3. Case 2 : Bank0 -> Bank 1
   1) Set HDSAR0, HDMAR0, HDSAR1, HDMAR1, HDTCR
   2) Set HDCON with 0x1000781
      (Match Interrupt Pending Enable, HDMA Interrupt Enable, Auto Load Enable, Aternate Enable, Select bank0 for current queuing bank, Enable Queuing Enable Bit 1, Run HDMA)

BLANK PAGE

21 Analog ASIC Interface Control

INTRODUCTION

The KS32C6400 has a block to interface with Analog ASIC(Mustang). This module is configured 16-bit shift register with 4bit counter.

SPECIAL REGISTER

Registers	Offset Address	R/W	Description	Reset Value
IDRVASIFR	0x5004	R/W	DRV_ASIF Data & Data output	0x00000000

31 ... 21	20	19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
	X	XXX	XXXXXXXXXXXXXXXX

[15:0] Output Data — Output data(16bit)

[19:16] Output Data Select — Output data select
"1" :2 -> "15" : 16

[20] Count Start — Count Start
"1" : start
"0" : count end

BLANK PAGE

22 RTC(Real Time Clock)

INTRODUCTION

Real Time clock(RTC) unit is operated by the sysem powr (+5V) or by the backup battery if the system power is turned off. The RTC transmits 8-bit data to the CPU as BCD (Binary-Coded Decimal) values using STRB/LDRB ARM operation. The data include second,minute,hour,date,day,month, and year. The RTC unit works with an external 32.768 KHz crystal.

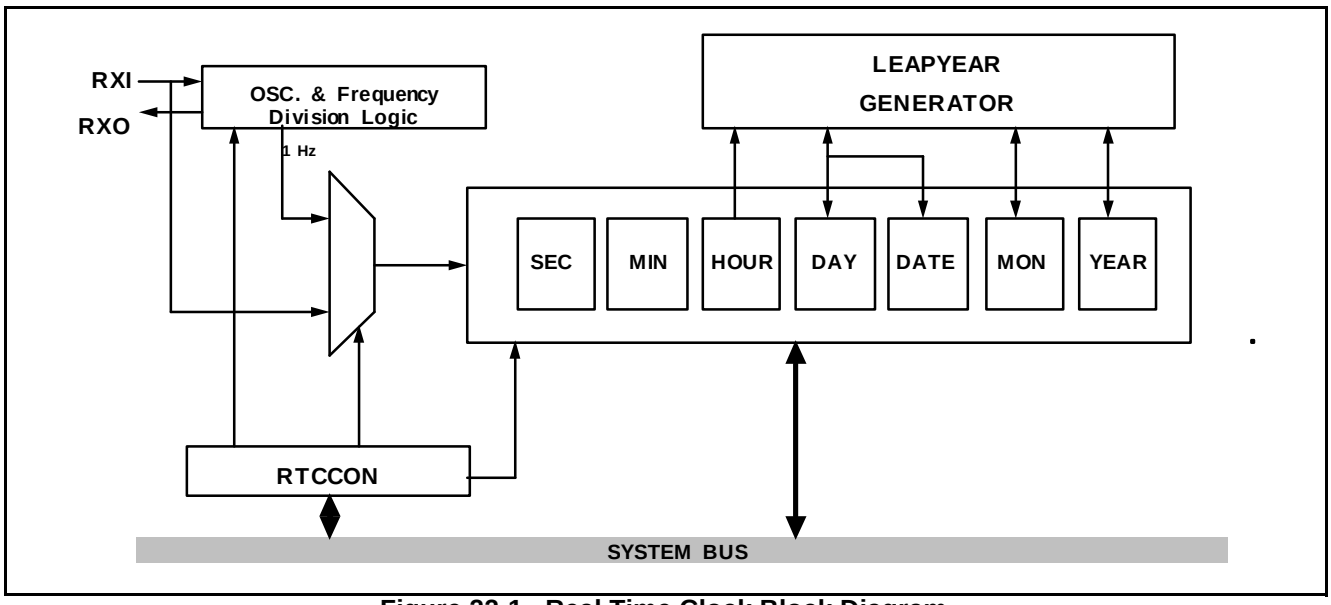


Figure 22-1. Real Time Clock Block Diagram.

LEAP YEAR GENERATOR

This caculates which the last date of each month is 28, 29, 30 or 31 that is based on data from BCDDAY, BCDMON and BCDYEAR. This also considers leap years in deciding the last date.
An 8-bit counter can just represent 2 BCD digits, so it cannot decide whether 00 year is a leap year or not. We know year 2000 is a leap year, therefore the leap year generator ishard-wired to work up to 2-29-00.
It is required to set bit 0 of the RTCCON register for interfacing between CPU and RTC logic. One second error can occur when the CPU reads or writes data into BCD counters and this can cause the change of the higher time units.When the CPU reads/writes data to/from the BCD counters, another time unit may be changed if BCDSEC register is overflowed.To avoid this problem,the CPU should reset BCDSEC register to 00h.The reading sequence of the BCD counters is BCDYEAR,BCDMON,BCDDATE,BCDDAY,BCDHOUR,BCDMIN and BCDSEC.It is required to read it again from BCDYEAR to BCDSEC if BCDSEC is zero.

BACKUP BATTERY OPERATION

The RTC logic is drived by a backup battery if the system powr is off.The interfaces of the CPU and RTC logic are blocked and the backup battery only drives the oscillation circuit and the BCD counters to minimize power dissipation.

RTCCON REGISTER

The RTCCON register is comprised of RTCE(RTC Enable: bit 0) which controls the write-disable of the BCD registers,RCLK(RTC Clock: bit1),CNTSEL(Counter Select: bit 2),and CLKRST(Clock Reset: bit 3) for testing.
Bit RTCE controls all interfaces between the CPU and the RTC,so it should be set to 1 in an initialization routine to enable data transfer after a system reset. Instead of working BCD with 1 Hz, bit RCLK enable the operation of BCD counters with an external clock which is entered through the pin RXI to the test BCD counters. Bit CNTSEL contverts the dependent operation of BCD counters into independent counters for testing. CLKRST resets the frequency divided-logic in the RTC unit.

Registers	Offset Address	R/W	Description	Reset Value
RTCCON	0xF840	R/W	RTC Control register	0x00000000

31	4	3	2	1	0
		X	X	X	X

[0] RTCE — RTC R/W Enable/Disable

0 = Disable
1 = Enable

[1] RCLK — BCD Clock Selection

0 = RXI/2^{15} divided clock
1 = RXI

[2] CNTSEL — BCD Count Selection

0 = Merge BCD counters
1 = Seperate BCD counters

[3] CLKRST — RTC Clock counter reset

0 = No reset
1 = Reset

BCDSEC COUNTER REGISTER

BCD count register for second.

Registers	Offset Address	R/W	Description	Reset Value
BCDSEC	0xF870	R/W	RTC Second register	Undef.

31	7	6	5	4	3	2	1	0
	COUNT VALUE							

[6:0] COUNT VALUE — Second Counting Value. (0 ~ 59)

BCDMIN COUNTER REGISTER

BCD count register for minute.

Registers	Offset Address	R/W	Description	Reset Value
BCDMIN	0xF874	R/W	RTC Minute register	Undef.

31	7	6	5	4	3	2	1	0
	COUNT VALUE							

[6:0] COUNT VALUE Minute Counting Value. (0 ~ 59)

BCDHOUR COUNTER REGISTER

BCD count register for hour.

Registers	Offset Address	R/W	Description	Reset Value
BCDHOUR	0xF878	R/W	RTC Hour register	Undef.

31	6	5	4	3	2	1	0
	COUNT VALUE						

[5:0] COUNT VALUE Hour Counting Value. (0 ~ 23)

BCDDAY COUNTER REGISTER

BCD count register for day.

Registers	Offset Address	R/W	Description	Reset Value
BCDDAY	0xF87C	R/W	RTC Day register	Undef.

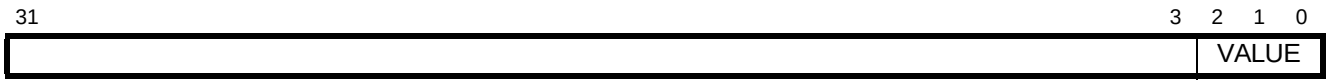
31	6	5	4	3	2	1	0
	COUNT VALUE						

[5:0] COUNT VALUE Day Counting Value. (0 ~ 28, 29, 30, 31)

BCDDATE COUNTER REGISTER

BCD count register for date.

Registers	Offset Address	R/W	Description	Reset Value
BCDDATE	0xF880	R/W	RTC Date register	Undef.

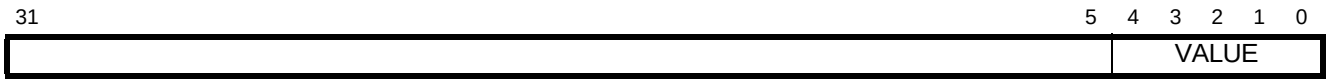


[2:0] COUNT VALUE Date Counting Value. (1 ~ 7)

BCDMON COUNTER REGISTER

BCD count register for month.

Registers	Offset Address	R/W	Description	Reset Value
BCDMON	0xF884	R/W	RTC Month register	Undef.

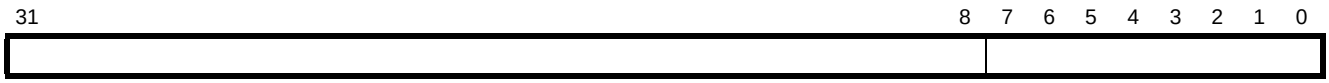


[4:0] COUNT VALUE Month Counting Value. (1 ~ 12)

BCDYEAR COUNTER REGISTER

BCD count register for year.

Registers	Offset Address	R/W	Description	Reset Value
BCDYEAR	0xF888	R/W	RTC Year register	Undef.



[7:0] COUNT VALUE Year Counting Value. (0 ~ 99)

23 POWER SAVE CONTROL

INTRODUCTION

This block provides the function which select the frequency of internal common CLOCK to decrease the power dissipation of Jupiter-II in sleep mode of system. Internal common clock is MCLK in normal mode and MCLK/20 in power save mode. RSTCLK is reset filtering logic.

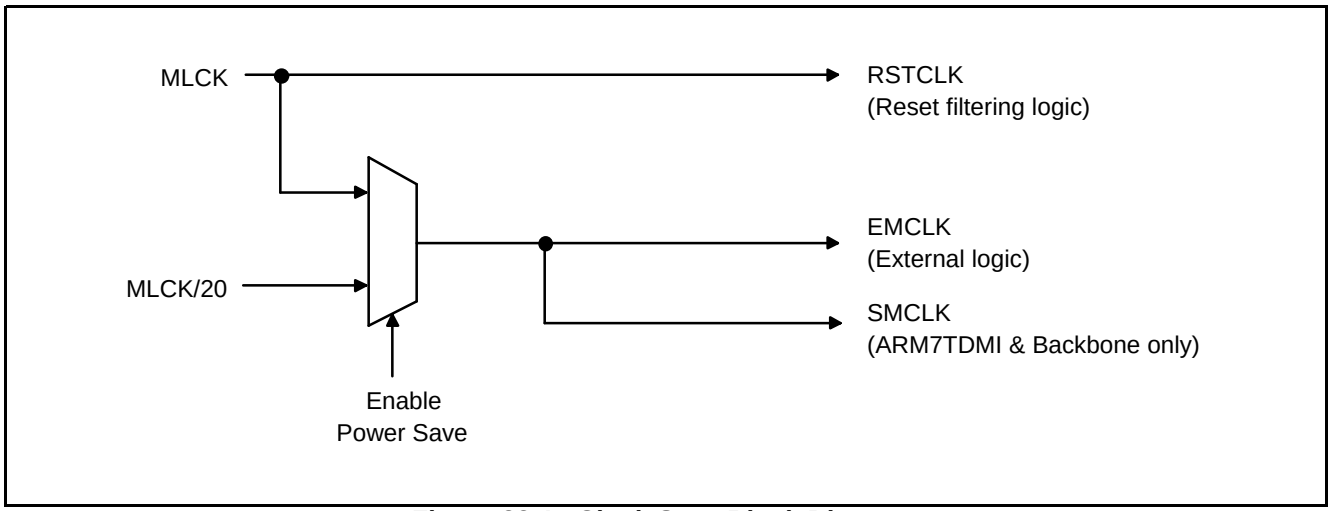


Figure 23-1. Clock Save Block Diagram.

EMCLK and SMCLK are selected MCLK or MCLK/20 by power save enable.

POWER SAVE CONTROL REGISTER

The PWRSAVCON register is comprised of ENPSAVE bit which decide to enable or not power save mode.

Registers	Offset Address	R/W	Description	Reset Value
PWRSAVCON	0x5820	R/W	Clock Save control register	0x00000000

31	1	0
		X

[0] ENPSAVE — Power Save Enable/Disable
0 = Disable
1 = Enable

BLANK PAGE

24 ELECTRICAL SPECIFICATION

ABSOLUTE MAXIMUM RATINGS

Table 24-1: Absolute Maximum Ratings (T_A = 25 °C)

Parameter	Symbol	Rating	Unit
Supply Voltage	V_{DD}	– 0.3 to + 7.0	V
Input Voltage	V_{IN}	– 0.3 to V_{DD} + 0.3	V
Operating Temperature	T_A	0 to + 70	°C
Storage Temperature	T_{STG}	– 40 to + 125	°C

THERMAL CHARACTERISTICS

Table 24-2: Thermal Characteristics (T_A = 25 °C)

Parameter	Symbol	Value	Unit
Thermal Impedance—Junction to Ambient Plastic 160-pin QFP	θ_{JA}	40	°C/W

D.C. ELECTRICAL CHARACTRERISTICS

Table 24-3: D.C. Electrical Characteristics (T_A = 0 °C to + 70 °C, V_{DD} = 4.75 V to 5.25 V)

Symbol	Parameter	Conditions	Min	Max	Unit
V_{IH}	Input High Voltage	TTL interface	2.0	-	V
		TTL schmitt trigger	-	2.1	
V_{IL}	Input Low Voltage	TTL interface	-	0.8	V
		TTL schmitt trigger	0.8	-	
I_{IH}	Input High Current	Input buffer, V_{IN} = V_{DD}	- 10	10	uA
I_{IL}	Input Low Current	Input buffer, V_{IN} = V_{SS}	- 10	10	uA
I_{LL2}		Input buffer with pull-up, V_{IN} = V_{SS}	- 200	- 10	
V_{OH}	Output High Voltage	Type 4, I_{OH} = - 4 mA (2)	2.4	-	V
		Type 8, I_{OH} = - 8 mA (2)			
V_{OL}	Output Low Voltage	Type 4, I_{OL} = 4 mA (2)	-	0.4	V
		Type 8, I_{OH} = 8 mA (2)			
I_{DD}	Quiescent Supply Current	V_{IN} = V_{SS} or V_{DD} , 33 MHz	-	180	mA

NOTES

1. We recommend for you to install the bypass capacitors between the V_{DD} and the V_{SS} of KS32C6400. The bypass capacitor increases the noise immunity of KS32C6400.
2. The pin type of O_1, O_2, O_3, I/O_1, I/O_4, and I/O5 is type 4, and I/O_3 is type 8.

Table 24-4: A.C. Electrical Characteristics (T_A = 0 °C to + 70 °C, V_{DD} = 4.75 V to 5.25 V)

Parameter	Symbol	Min	Max	Unit
RESET Pulse Width	tRST	69	–	MCLK
CLKOUT Rising Time from MCLK	tCKOUTR	5.0	15.4	ns
CLKOUT Falling Time from MCLK	tCKOUTF	4.8	14.5	ns
nEINT1,2 Setup before MCLK	tINTS	0	–	ns
nEINT1,2 Hold after MCLK	tINTH	5.0	–	ns
nEINT1,2 Pulse Width	tINTW	3.0	–	MCLK
Parallel Port Input Hold after MCLK	tPINH	0	27	ns
Parallel Port Output Valid from MCLK	tPOV	3	17	ns
PPD[7:0] Valid from MCLK	tPDV	11	23	ns
PPD[7:0] High Impedance from MCLK	tPDZ	4	13	ns
nXDREQ Setup Time before MCLK	tXDREQS	6	–	ns
nXDREQ Hold Time before MCLK	tXDREQH	4	–	ns
nXDREQ Pulse Width	tXDREQW	2	–	MCLK
nXDACK Valid from MCLK	tXDACK	4	29	ns
nXDACK Pulse Width	tXDACKW	18	–	MCLK
Address Hold Time	tADDRH	7.1	–	ns
Address Delay time	tADDRD	–	25.1	ns
ROM Bank Chip Select Delay Time	tNRCS	–	20.6	ns
ROM/SRAM/Extra I/O Bank Out Enable Delay	tNROE	–	23.5	ns
SRAM/Extra I/O Bank Write Enable Delay	tNRWE	–	18.2	ns
SRAM/Extra I/O Bank Write Byte Enable Delay	tNWBE	–	18.1	ns
Read Data Hold Time	tRDH	3.0	–	ns
Write Data Delay Time (SRAM/Extra I/O)	tWDD	–	9.8	ns
Write Data Hold Time (SRAM/Extra I/O)	tWDH	26.3	–	ns
DRAM Row Address Strobe Active Delay	tNRASF	–	15.2	ns
DRAM Row Address Strobe Release Delay	tNCASR	–	27.0	ns
DRAM Column Address Strobe Active Delay	tNCASF	–	16.1	ns
DRAM CAS Signal Release Delay Time	tNCASR	–	17.1	ns
DRAM CAS Write Active Delay	tNCASW	–	19.8	ns
DRAM Bank Write Enable Delay Time	tNDWE	–	24.4	ns
DRAM Bank Out Enable Delay Time	tNDOE	–	23.5	ns
External I/O Bank Chip Select Delay Time	tNECS	–	20.6	ns
Special I/O Bank Read Signal Delay Time	tNIORD	–	23.5	ns
Speical I/O Bank Write Signal Delay Time	tNIOWR	–	18.2	ns
DRAM Write Data Delay Time (DRAM)	tWDDD	–	14.2	ns
DRAM Write Data Hold Time (DRAM)	tWDDH	7.4	–	ns

TIMING DIAGRAM

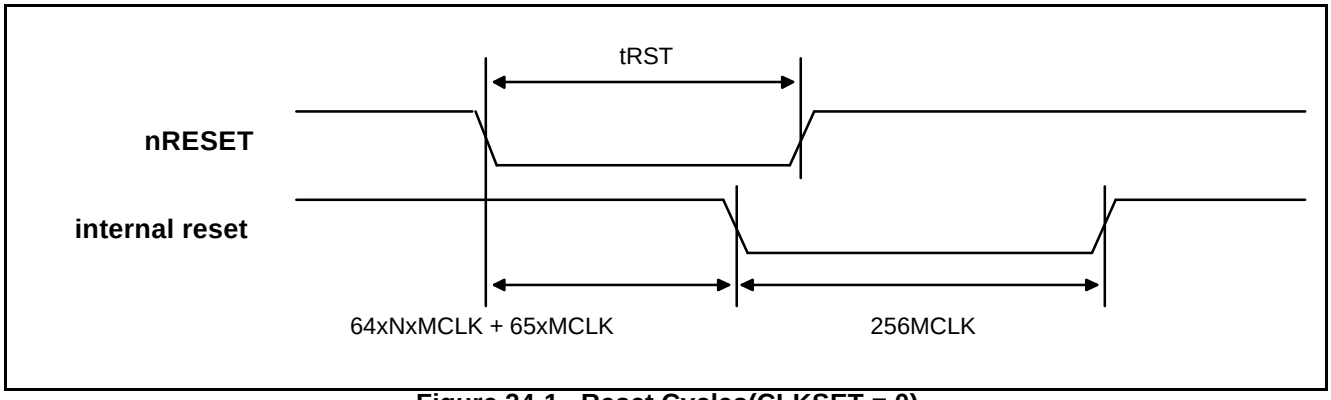


Figure 24-1. Reset Cycles(CLKSET = 0)

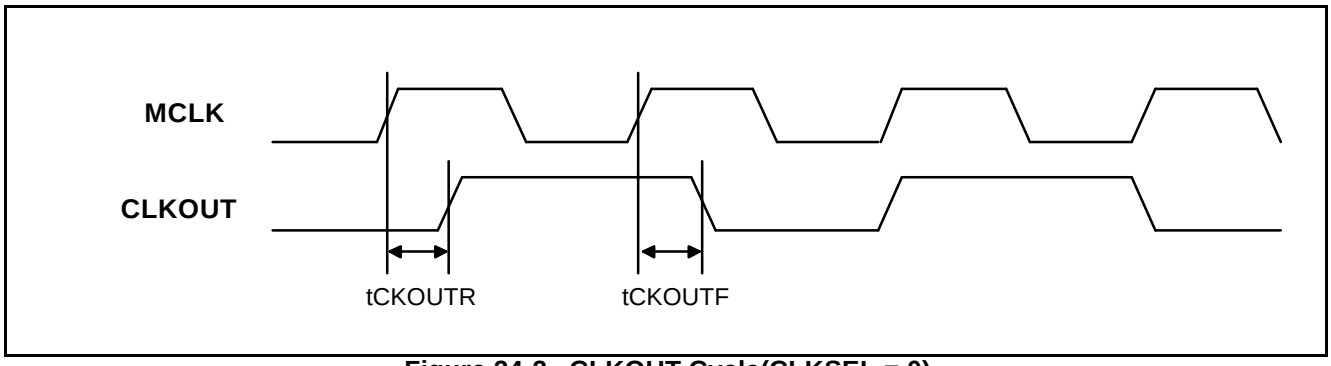


Figure 24-2. CLKOUT Cycle(CLKSEL = 0)

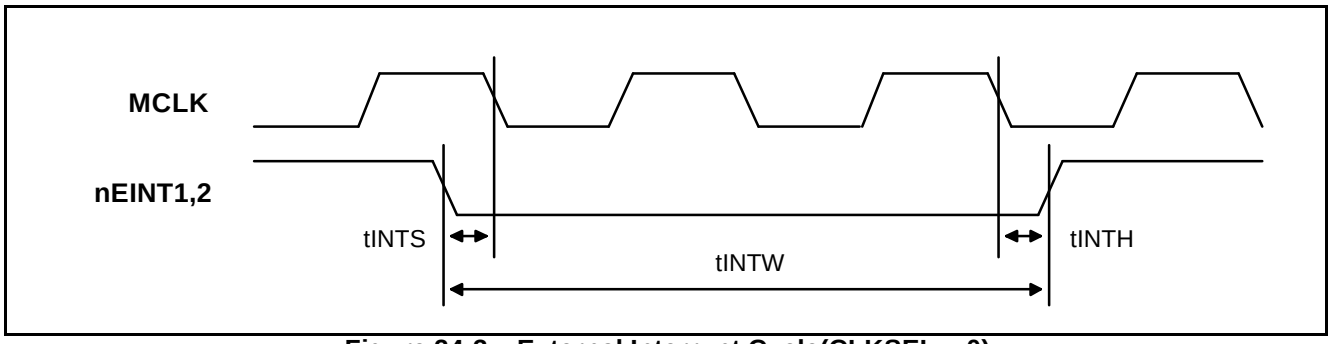


Figure 24-3. External Interrupt Cycle(CLKSEL = 0)

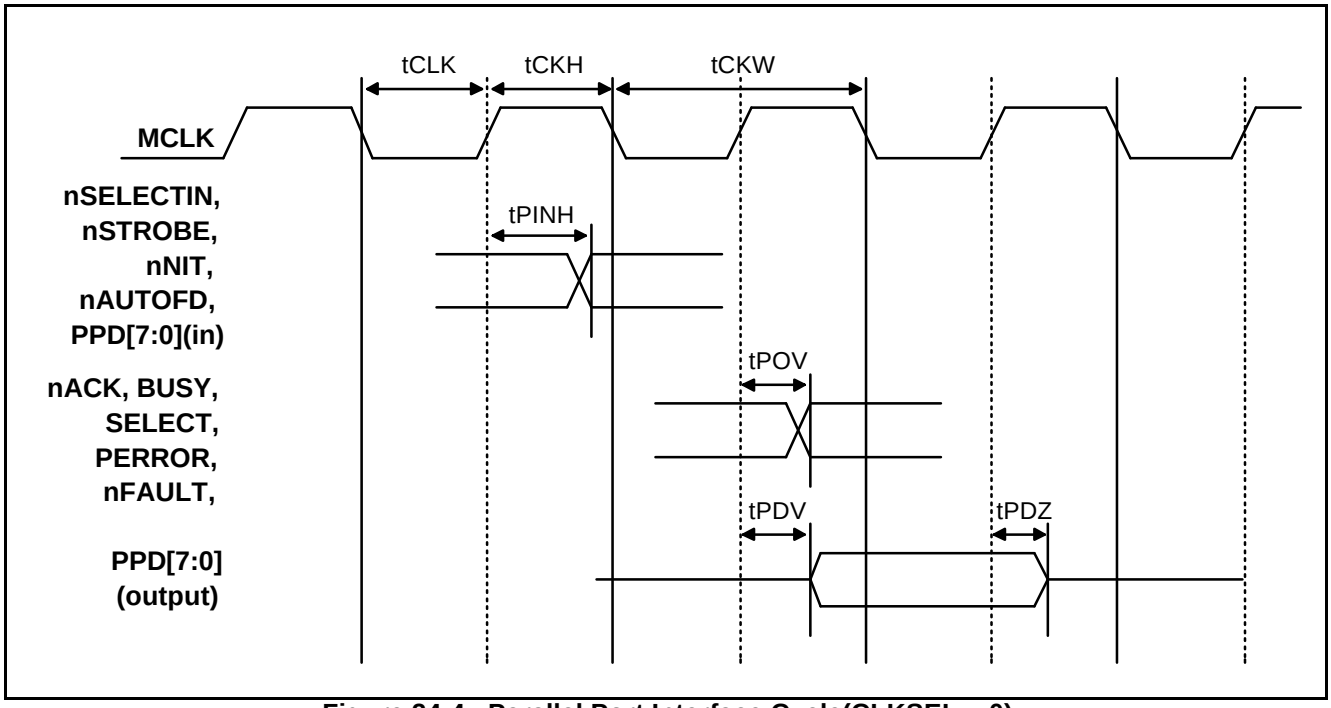


Figure 24-4. Parallel Port Interface Cycle(CLKSEL = 0)

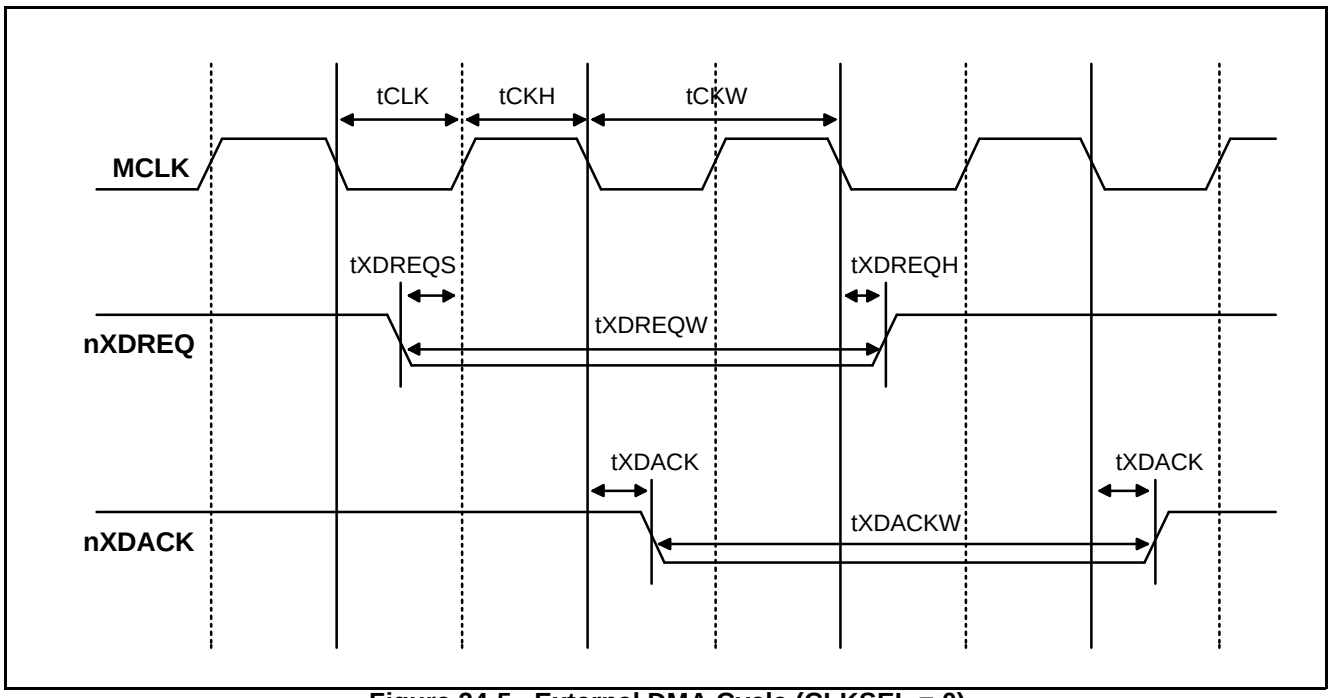


Figure 24-5. External DMA Cycle (CLKSEL = 0)

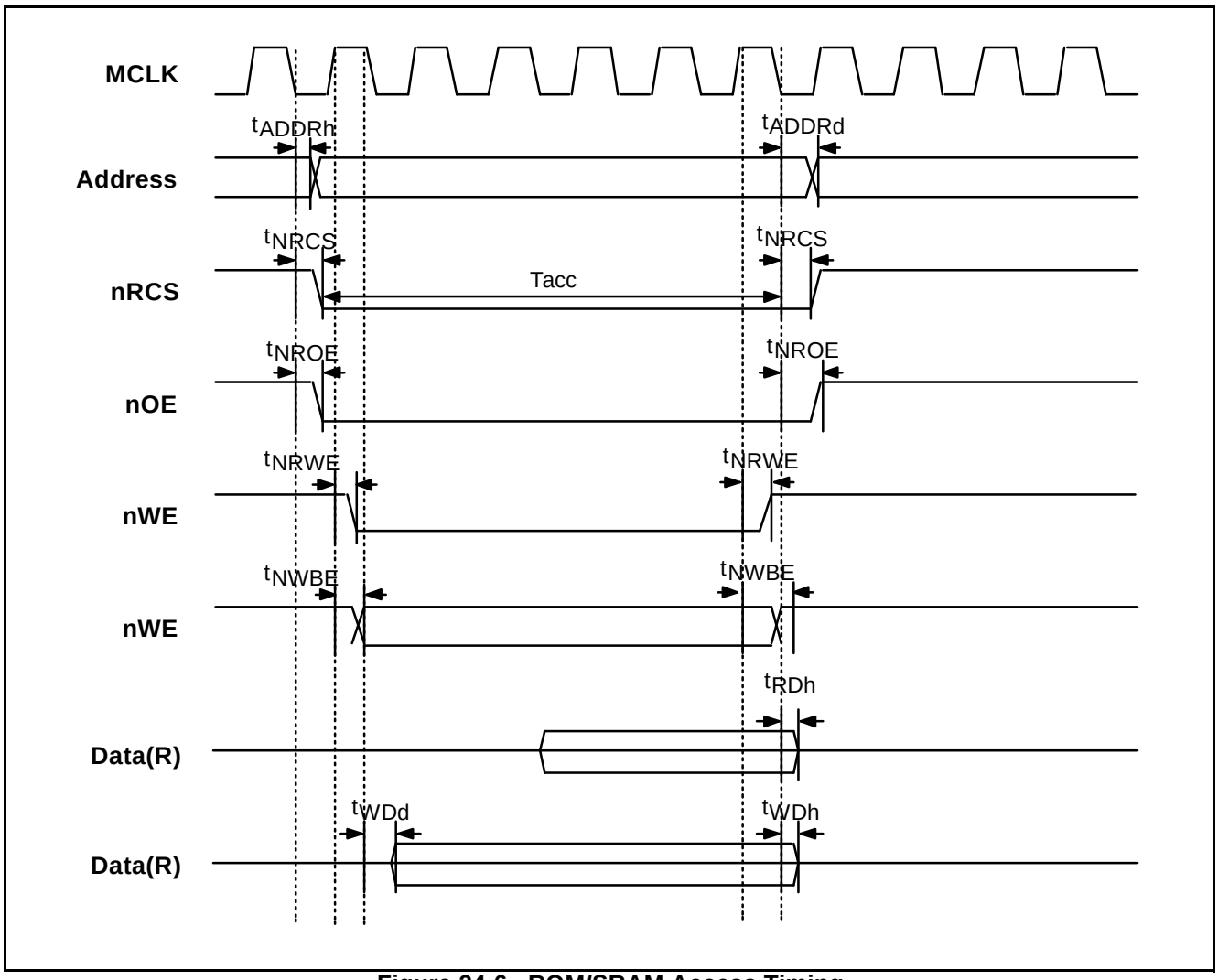


Figure 24-6. ROM/SRAM Access Timing

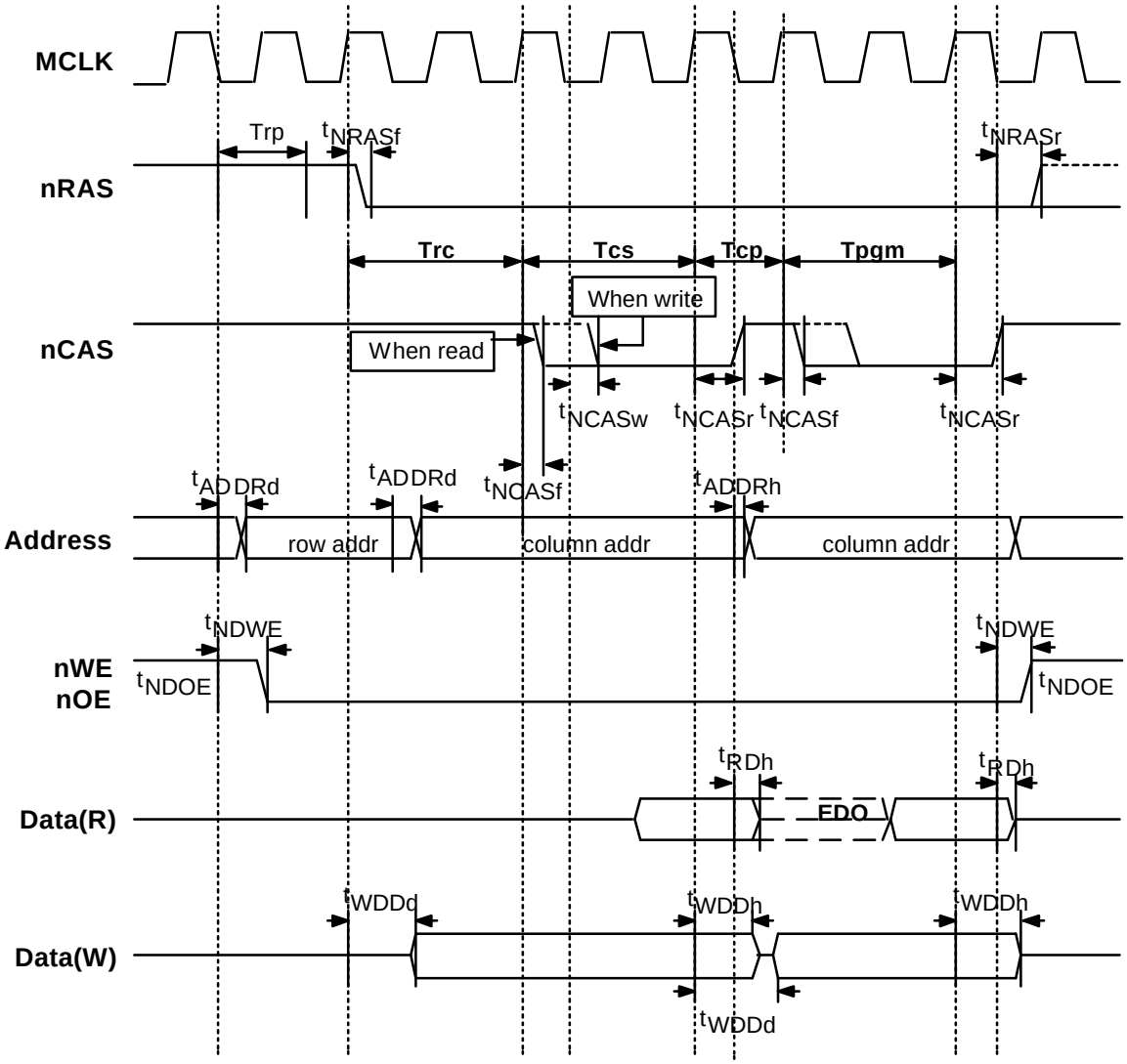


Figure 24-7. DRAM Bank Access Timing

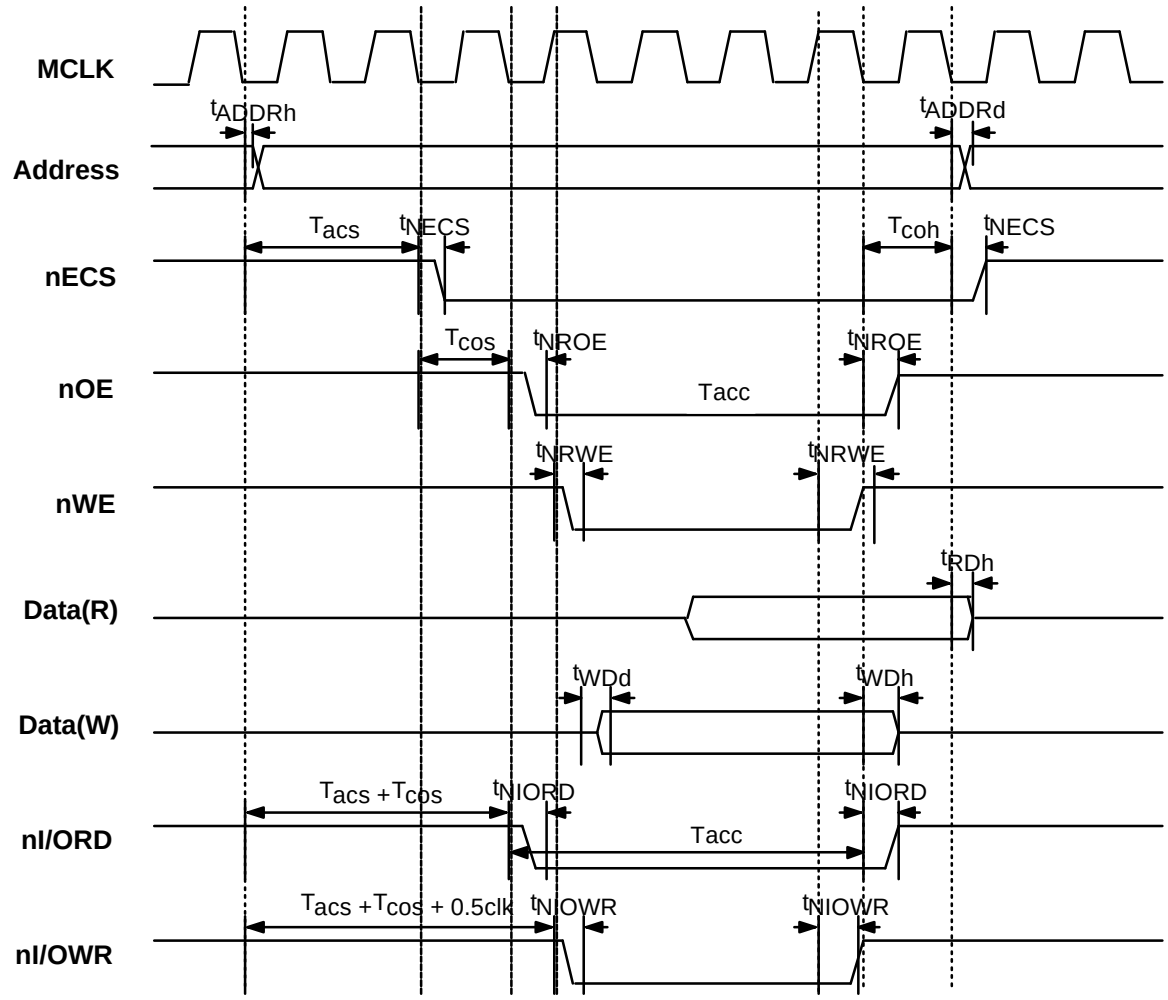


Figure 24-8. Extra Bank Access Timing

25 MECHANICAL SPECIFICATION

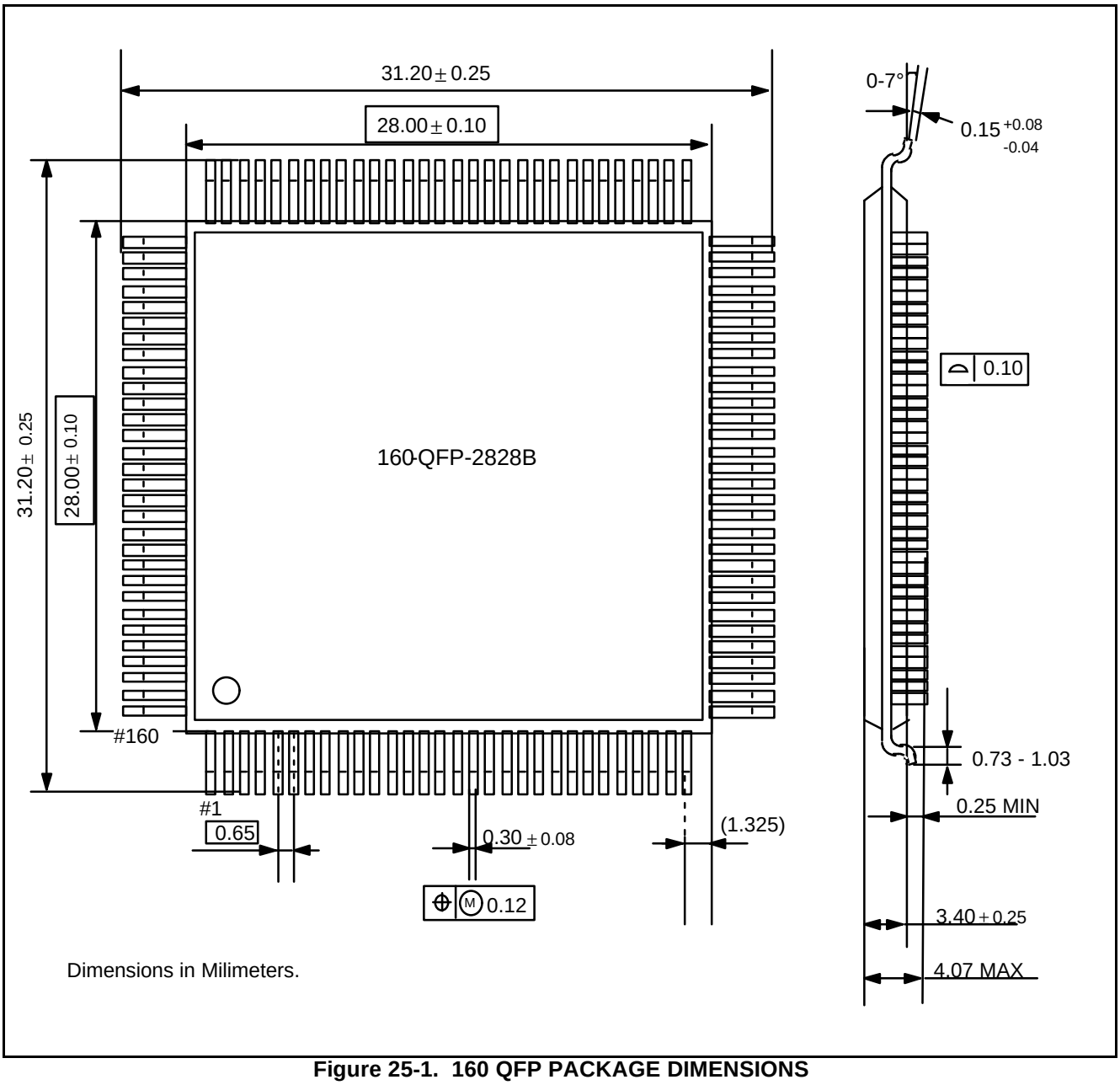


Figure 25-1. 160 QFP PACKAGE DIMENSIONS